MAX-PLANCK-INSTITUT FÜR SOZIALRECHT UND SOZIALPOLITIK
MAX PLANCK INSTITUTE FOR SOCIAL LAW AND SOCIAL POLICY

mea Munich Center for the Economics of Aging

Continuous-time speed for discrete-time models: A Markov-chain approximation method

Ivo Bakota, Matthias Kredler

01-2022

MEA DISCUSSION PAPERS

mea – Amalienstr. 33_D-80799 Munich_Phone +49 89 38602-355_Fax +49 89 38602-390_www.mpisoc.mpg.de

Continuous-time speed for discrete-time models: A Markov-chain approximation method

Ivo Bakota, Matthias Kredler

Abstract:

We propose a Markov-chain approximation method for discrete-time control problems, showing how to reap the speed gains from continuous-time algorithms in this class of models. Our approach specifies a discrete Markov chain on a grid, taking a first-order approximation of conditional distributions in their first and second moments around a reference point. Standard dynamic-programming results guarantee convergence. We show how to apply our method to standard consumption-savings problems with and without a portfolio choice, realizing speed gains of up to two orders of magnitude (a factor 100) with respect to state-of-the-art methods, when using the same number of grid points. This is without significant loss of precision. We show how to avoid the curse of dimensionality and keep computation times manageable in high-dimensional problems with independent shocks. Finally, we show how our approach can substantially simplify the computation of dynamic games with a large state space, solving a discrete-time version of the altruistic savings game studied by Barczyk & Kredler (2014).

Zusammenfassung:

Wir schlagen eine Markov-Ketten-Approximationsmethode für zeitdiskrete Steuerungsprobleme vor und zeigen, wie man die Geschwindigkeitsvorteile von zeitstetigen Algorithmen in dieser Modellklasse nutzen kann. Unser Ansatz spezifiziert eine diskrete Markov-Kette auf einem Gitter, wobei eine Approximation erster Ordnung der bedingten Verteilungen in ihren ersten und zweiten Momenten um einen Referenzpunkt herum verwendet wird. Standardergebnisse der dynamischen Optimierung garantieren Konvergenz. Wir zeigen, wie unsere Methode auf kanonische Sparprobleme mit und ohne Portfoliowahl angewandt werden kann, wobei Geschwindigkeitsgewinne von bis zu zwei Größenordnungen (ein Faktor 100) im Vergleich zu modernsten Methoden erzielt werden, wenn dieselbe Anzahl von Gitterpunkten verwendet wird. Dies geschieht ohne signifikanten Verlust an Präzision. Wir zeigen, wie man den Fluch der Dimensionalität vermeidet und die Berechnungszeiten bei hochdimensionalen Problemen mit unabhängigen Schocks überschaubar hält. Schließlich zeigen wir, wie unser Ansatz die Berechnung von dynamischen Spielen mit einem großen Zustandsraum erheblich vereinfachen kann, indem wir eine zeitdiskrete Version des altruistischen Sparspiels lösen, das in Barczyk & Kredler (2014) untersucht wurde.

Keywords:

Markov-chain approximation (MCA) methods; altruism; tensors; controlled Markov chains

JEL Classification:

C60, C61, C73, E21, G11

Continuous-time speed for discrete-time models: A Markov-chain approximation method*

Ivo Bakota°
Munich Center for the Economics of Aging
Max Planck Institute for Social Law and Social Policy

Matthias Kredler•
Universidad Carlos III de Madrid

Abstract

We propose a Markov-chain approximation method for discrete-time control problems, showing how to reap the speed gains from continuous-time algorithms in this class of models. Our approach specifies a discrete Markov chain on a grid, taking a first-order approximation of conditional distributions in their first and second moments around a reference point. Standard dynamic-programming results guarantee convergence. We show how to apply our method to standard consumption-savings problems with and without a portfolio choice, realizing speed gains of up to two orders of magnitude (a factor 100) with respect to state-of-the-art methods, when using the same number of grid points. This is without significant loss of precision. We show how to avoid the curse of dimensionality and keep computation times manageable in high-dimensional problems with independent shocks. Finally, we show how our approach can substantially simplify the computation of dynamic games with a large state space, solving a discrete-time version of the altruistic savings game studied by Barczyk & Kredler (2014).

Keywords: Markov-chain approximation (MCA) methods; altruism; tensors; controlled Markov chains
JEL codes: C60, C61, C73, E21, G11

---

* This version: May, 2022.

° bakota@mea.mpisoc.mpg.de
• Email: matthias.kredler@uc3m.es

1 Introduction

Motivation. There has been a recent, renewed interest in continuous-time methods in economics. This is not only for their theoretical tractability, but also for the ease and speed of computation.[1] Computation of these models is usually performed by Markov-chain-approximation (MCA) methods or variants thereof.[2] In a nutshell, this paper applies the same principles to create a MCA method for discrete-time problems, thus harnessesing the speed gains and tractability of continuous-time algorithms also in discrete-time models (which are more commonly used in economics).

We now describe briefly the **state-of-the-art** and sketch how **our approach** differs. The standard solution method for discrete-time stochastic control problems in economics (see, among others, Heer & Maußner (2011)) is as follows: a) Discretize continuous state variables on a grid, b) discretize shocks on a finite support following Tauchen's (1986) method, c) interpolate value (or policy) functions to calculate continuation values on the points sampled in the previous step, d) solve for the best policy given these continuation values, and e) update the value (or policy) function. Note here that when computing continuation values in Step c), the method essentially amounts to computing a weighted combination of function values at neighboring grid points — or a weighted sum in the case of the most popular method, linear interpolation. At this point, our first modification (the *modified Tauchen step*) comes in: Instead of keeping shock discretization and interpolation separate, we *directly* specify a probability distribution over grid points, the weights being a smooth function of the controls. This sets up a controlled Markov chain on the (fixed) grid. Since the resulting control problem is usually still untractable, the second step —and key novelty— of our approach comes in: *linear moment approximation*. We take a first-order approximation of the distribution vector over grid points in the 1st and 2nd moments of the stochastic process. Doing so, we obtain simple first-order conditions (FOCs) in most of our applications. If FOCs are still unwieldy, in a third step one may apply *law-of-motion/return-function approximation*, taking a first-order Taylor expansion

[1] A recent prominent example is Achdou et al.'s (2022) analysis of the heterogeneous-agents incomplete-markets model.

[2] Note here that finite-difference schemes can be framed as a sub-case of MCA. The standard reference on MCA methods is Kushner et al. (2001). See also Phelan & Eslami (2021), who provide an excellent introduction to MCA methods in macroeconomics.

of the mapping from control variables to moments and/or the return function.. Finally, we turn to high-dimensional examples in which shocks are independent across the different dimensions; using tensor algebra we show how to avoid the curse of dimensionality and accelerate our baseline algorithm by several orders of magnitudes.

Our approach is inspired by **continuous-time MCA methods**, which proceed in the following steps: i) Discretize state variables on a grid, ii) construct a (discrete-time) Markov chain on this grid that matches the first and second moments of the underlying diffusion process (usually restricting jumps to a close neighborhood), thereby iii) dropping lower-order terms in laws of motion and iv) making the time increment small enough so that all transition probabilities are positive (which is the stability condition).

We use three **examples** to illustrate our approach, comparing our method to the state-of-the-art in speed and precision. We first solve a standard consumption-savings model in Section 2, in which time savings of our algorithm are about one order of magnitude (a factor 10). Second, we solve a portfolio problem with two assets in Section 3, where speed gains are on the order of a factor 100. In both these examples, our algorithm yields solutions that are very close to those from a state-of-the-art algorithm and to a benchmark with a closed-form solution, suggesting that our algorithm sacrifices very little (if anything) on precision. Third, we apply our algorithm to a high-dimensional example with two decision makers in Section 4: a dynamic savings game played by two savers, a parent and a child, the parent being altruistic towards the child and providing transfers. The general recipe for our algorithm is described in Appendix A. Appendix B introduces concepts from tensor algebra that compactly describe operations on multi-dimensional arrays and shows how to speed up computations under independent shocks across multiple dimensions.[3]

Our approach has the following **advantages**, many of which it shares with continuous-time MCA algorithms. First, it leads to simple FOCs (often independent ones when multiple controls are present) and thus avoids costly root-finding routines or non-linear solvers. Second, it casts laws of motion as sparse matrices, thus allowing for fast computation also in high-level programming languages such as *Matlab* and *Julia* (which both have built-in sparse matrix routines that are optimized for speed). Multiplication by sparse matrices speeds up both backward operations (calculation of continuation values and other expec-

[3]See the functions `MatrixTimesArray.m` and `kronm.m` that we provide on Matlab Central.

tations) and forward operations (evolution of distributions). Third, results from the literature on controlled Markov chains apply to our algorithm, thus guaranteeing convergence of the algorithm by standard dynamic-programming arguments; there are results available proving monotonicity and a rapid speed of convergence.[4] To apply these results in our approach, one has to ensure that all transition probabilities in the approximating Markov chain are non-negative; we provide simple conditions to ensure this here, for both low- and high-dimensional problems. Fourth, our approximation scheme allows us to solve dynamic *games* (i.e. settings with more than one decision maker) that are otherwise untractable. In the example in Section 4, we show how to adapt our algorithm to solve a discrete-time version of Barczyk & Kredler's (2014) model of altruistically-linked savers. We show that our algorithm yields the same qualititative results as theirs, at a computation time of about one second for the value-function-iteration loop with a state space of size $60 \times 60 \times 2 \times 2$. This makes computation of large-scale savings games feasible also in discrete time.[5]

However, our approach also has some **drawbacks** that the user should be aware of;. First, to reap the full benefits of our approach, one has to ensure that the state does not move more than one grid point up or down in expectation in each period (and in each continuous dimension). In our applications, this can always be achieved by choosing the time period short enough so that this is ensured for any fixed grid. Second, there is a loss of precision in approximation steps 1-3. However, one has to bear in mind that also alternative measures make approximations. e.g. when interpolating functions. In our examples, there is no visible loss of accuracy when compared to the state-of-the-art; our approach in fact tends to yield solutions that are closer to closed-form solutions of germane benchmark models and has comparable or even slightly smaller Euler equation residuals.[6]

[4]Puterman & Brumelle (1979), for example, establish monotonicity and a quadratic speed of convergence near the solution for policy-function-iteration algorithm for a controlled finite-state Markov chain. As Phelan & Eslami (2021) find for continuous-time settings, we also find that modified policy function iteration (applying a policy guess for a finite number k of periods when updating the value function) is the fastest algorithm (classical policy function iteration can be seen as setting $k = \infty$). We are not aware of similar results for the state-of-the-art approach using interpolation.

[5]Fifth, our approach can easily accommodate non-linear grids, which allows to set grid points economically.

[6]Third, for the convergence results on controlled Markov chains to apply, one has to reflect back stochastic processes at the "artificial grid margins", i.e. at the grid boundaries that are not part of the model's physical environment, but are imposed by the finiteness of computer memory. This reflection distorts value and policy functions close to the grid boundaries. Thus, the user should always ensure that results close to these bound-

Our approach connects to various methods in the **literature**. First, our approach nests two algorithms that are routinely employed: a) value-function iteration with linear interpolation obtains as a limit of our algorithm; this is the case in one-dimensional problems with a deterministic law of motion, such as in a standard consumption-savings model; b) a standard trinomial grid method (as is routinely used to approximate standard continuous-time diffusions) obtains when choosing a three-point approximating distribution in our approach. Second, our approach yields (approximate) Bellman Equations that are very similar to the Hamilton-Jacobi-Bellman equations resulting from (continuous-time) stochastic calculus, which is a popular tool especially in the finance literature[7], but also increasingly in macroeconomics. This analogy obtains since our approach focuses on how controls affect the conditional mean and variance of the continuous state variables, which is exactly what stochastic calculus does. However, our discrete-time setting has the advantages that it is i) arguably more intuitive, ii) closer to the original setup of most models in economics, and that iii) it avoids the mathematical intricacies of Brownian Motion.

For an overview of methods used to solve DSGE models and a general description of finite-element methods, which the Markov-chain approximation is akin to, see **?**. Other standard references for the benchmark algorithms that we compare our approach to are **?** and **?**. **?** also contains a treatment of discrete-state Markovian control problems with discrete choices; these are distinct from our approach, however, since i) the underlying physical environment features continuous states in our problem and ii) our approach features continuous choice variables, even in the approximated economy. **?** also use tensor notation, but for a different purpose than we do: they aim to simplify the notation of polynomials that come in up in function approximation, whereas we do so to simplify the computation of transition matrices.

aries do not affect results by ensuring that the equilibrium process spends negligible time close to artificial boundaries. We also remark here that other methods, such as extrapolation, can lead to similar distortions at artificial grid margins, but without the benefit of added stability.

[7] See Karatzas et al. (1998) for an introduction to continuous-time methods in finance.

2 Consumption-savings model

2.1 Setup

Consider an infinitely-lived saver who orders consumption streams $\{c_t\}_{t=0}^{\infty}$ by the criterion

$$\mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t u(c_t), \tag{1}$$

where $\beta \in (0,1)$ and where we assume the CRRA utility functional $u(c_t) = \frac{c^{1-\gamma}}{1-\gamma}$ with $\gamma > 0$. The law of motion for assets a_t is

$$a_{t+1} = (a_t - c + y)\tilde{R}_{t+1}, \tag{2}$$

where initial assets $a_0 \geq 0$ and income $y \geq 0$ is given. The saver faces a no-borrowing constraint, i.e. $a_{t+1} \geq 0$. The return $\tilde{R}_{t+1}$ is distributed i.i.d. log-normally with variance σ and mean r, thus the density conditional on choosing savings $s_t \in [0, a_t + y]$ is $a_{t+1} \sim f(a_{t+1}|s_t) = \ln \mathcal{N}(r + \ln s_t, \sigma)$. The Bellman Equation that characterizes the saver's value function $W(\cdot)$ is captured by the operator T defined by

$$(TV)(a) = \max_{s \in [0, a+y]} \Big\{ u(a + y - s) + \beta \mathbb{E}_{\tilde{R}'} V(\tilde{R}' s) \Big\}. \tag{3}$$

The solution to the problem is the unique fixed point $W = T(W)$ of this operator. It can be approximated by successive iterations, $W_{n+1} = TW_n$, on any starting guess W_0, as is well-known.

2.2 Algorithms

State of the art. We will first review the standard method of approximating a solution to the Belllman Equation (3) and use this to motivate our approach. In iteration $n+1$, a guess $W_n(\cdot)$ of the value function is given on a finite set of grid points. The canonical method first discretizes the shock $\tilde{R}'$ via Tauchen's (1986) method, yielding a discrete set $\{\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_I\}$ of states with associated probabilites $\{p_1, \ldots, p_l\}$. It then evaluates the

continuation (value) function $W_n(\cdot)$ once for each possible realization of the discrete shock, computing $\sum_{i=0}^{I} p_i W_n(\tilde{x}_i)$. To illustrate, assume that we use linear interpolation and that all discretized-shock realizations fall in between two grid-points (call them $\tilde{X}_l$ and $\tilde{X}_h$). The standard method will then evaluate $W_n(\cdot)$ on the neighboring two grid points I times, linearly interpolating $V(\tilde{x}_i) = V(\tilde{X}_l)\frac{\tilde{X}_h - x_i}{\tilde{X}_h - \tilde{X}_l} + V(\tilde{X}_h)\frac{\tilde{x}_i - \tilde{X}_l}{\tilde{X}_h - \tilde{X}_l}$, where the weight given to the two grid points is given by their linear distance from the point $\tilde{x}_i$.[8] The continuation value is then computed by summing all the possible realizations weighted by their probabilities p_i: $CV = \sum_{i=0}^{I} p_i (V(\tilde{X}_l)\frac{\tilde{X}_h - \tilde{x}_i}{\tilde{X}_h - \tilde{X}_l} + V(X_h)\frac{\tilde{x}_i - \tilde{X}_l}{X_h - X_l})$, where we see that both $V(\tilde{X}_l)$ and $V(\tilde{X}_h)$ are evaluated I times. Our idea now is the following: Instead of sampling shocks and then interpolating in two separate steps, we *directly* specify weights on grid points, which we will let vary smoothly in the agent's savings choice. In the simple example, we can construct weights $\tilde{w}_i$ actually construct weights collecting terms as follows:

$$CV = \underbrace{\sum_{i=0}^{I} p_i \frac{\tilde{X}_h - \tilde{x}_i}{\tilde{X}_h - \tilde{X}_l}}_{\tilde{w}_l} V(\tilde{X}_l) + \underbrace{\sum_{i=0}^{I} p_i \frac{\tilde{x}_i - \tilde{X}_l}{\tilde{X}_h - \tilde{X}_l}}_{\tilde{w}_h} V(\tilde{X}_h). \tag{4}$$

The advantage is that once having constructed the weights ω_l and ω_h, we only have to evaluate the value function at the neighboring grid points once.

When moving to more complex examples (with Tauchen shocks spanning more than one grid interval and when using non-linear interpolation), then the weights under the canonical method become more complex, non-smooth functions of $\{p_i, \tilde{x}_i, \}$ etc. and their combination occurs not necessarily a simple sum. But the principle stays the same: the canonical method uses information from a small number of neighboring grid points, i.e. function values $\{W_n(\tilde{X}_j)\}$ for some subset of grid points $\{\tilde{X}_j\}_{j=1}^{m}$, to approximate a continuation value, CV. Our idea is to streamline this procedure, specifying a direct mapping from controls (here: savings s) to a set of weights on grid points that i) captures the shock's probability distribution, but that is also ii) smooth and iii) tractable when it comes to calculating first-order conditions.

Modified Tauchen step. To specify our weights, we proceed in the way of of Tauchen's

[8] Note here that interpolating several times is computationally even more expensive for non-linear interpolation.

original paper, i.e. by calculating the probability that a random variable falls into a bin corresponding to a discrete grid point. However, we use as a grid the algorithm's asset grid, $\{\hat{x}_j\}_{j=1}^N$, and do not create an additional shock grid $\{\tilde{x}_i\}$, as the canonical method does. Setting the bin boundaries half-way between grid points, we calculate the probability weight on grid point $\hat{x}_j$ (given a savings choice s) as

$$\omega_j(s) = \int_{(\hat{x}_j+\hat{x}_{j-1})/2}^{(\hat{x}_j+\hat{x}_{j+1})/2} f(a|s)da = F((\hat{x}_j + \hat{x}_{j+1})/2|s) - F((\hat{x}_j + \hat{x}_{j-1})/2|s), \quad (5)$$

where we define $F(a|s) = [f(x|s)]_{-\infty}^{a}$ as the cumulative distribution function (cdf) of the log-normal density over assets given s. For practical reasons, we set small weights below some tolerance level ϵ to zero.[9]

Linear moment approximation step. Taking a glance at the weights from (5), it is clear that the modified Tauchen step per se does not lead to a much more tractable problem. When calculating a first-order condition, one would have to take the derivative of the weights in the choice s; this is feasible, but is cumbersome both analytically and computationally. When inspecting the distributions of assets for quantitatively reasonable savings choices (at least at a yearly model frequency), one observes that their shape remains quite similar. This motivates our second, and key, step: *linear moment approximation*. The idea is to approximate linearly how the conditional distribution over the asset grid (a vector) changes as one moment (here: s) changes. In order to obtain proper probability distributions, we will do so by mixing two distributions. A natural choice is to use the distributions that obtain from choosing savings $s = \hat{a}_n$ and $s = \hat{a}_{n+1}$ exactly on two neighboring grid points, here for a saver choosing savings $s \in (\hat{a}_n, \hat{a}_{n+1})$. We will then choose the grid and time interval such that the equilibrium savings choice is likely not to move the state by more than one grid point at a time, i.e. so that $s \in [\hat{a}_{n-1}, \hat{a}_{n+1}]$. This then leads to a linear

[9] In general, in this modified Tauchen step the user has to specify a set of moments that parameterize distribution that captures the law of motion. The default choice for these moments are the mean and the variance, see Appendix A for the reasons. In this simple example we use only one variable (s) to parameterize the conditional distribution. Note that s mainly increases the mean of the log-normal, but a higher s also increases the variance of a' (for constant σ); intuitively, the same proportional shock means a higher absolute movement in assets at higher savings. The effect on the variance is secondary here and we obtain good approximations with only one parameter. In general, it should always be checked if approximating distributions are good (in terms of mean and variance) when using different parameters for the conditional distribution.

Taylor expansion around the reference distribution $f(\cdot|s = a_n)$ — which arises from zero savings—, using directional gradients of (forward if $s > a_n$ and backward if $s < a_n$).[10] We will now see how this comes about.

We now describe this procedure in matrix notation. First, for each grid point a_n on the N-size grid, we construct a 1-by-N row vector $\mathbf{f}^{(0)}(a_n)^T$ that discretizes the conditional distribution given savings $s = a_n$, collecting the weights from the modified Tauchen step (5). We then collect all N distributions in the N-by-N matrix $\mathbf{F}^{(0)} = [\mathbf{f}^{(0)}(a_1); \mathbf{f}^{(0)}(a_2); \dots]$; we make this matrix sparse by setting elements below some threshold ϵ to zero. For savings $s \in (a_n, a_{n+1})$, we then approximate

$$\mathbf{f}(s) \simeq \frac{a_{n+1} - s}{a_{n+1} - a_n}\mathbf{f}^{(0)}(a_n) + \frac{s - a_n}{a_{n+1} - a_n}\mathbf{f}^{(0)}(a_{n+1}) \tag{6}$$

$$= \mathbf{f}^{(0)}(a_n) + [\underbrace{s - a_n}_{=m \text{ (drift)}}][\underbrace{(\mathbf{f}^{(0)}(a_{n+1}) - \mathbf{f}^{(0)}(a_n))/(a_{n+1} - a_n)}_{\equiv \mathbf{d}_m(a_n)}], \tag{7}$$

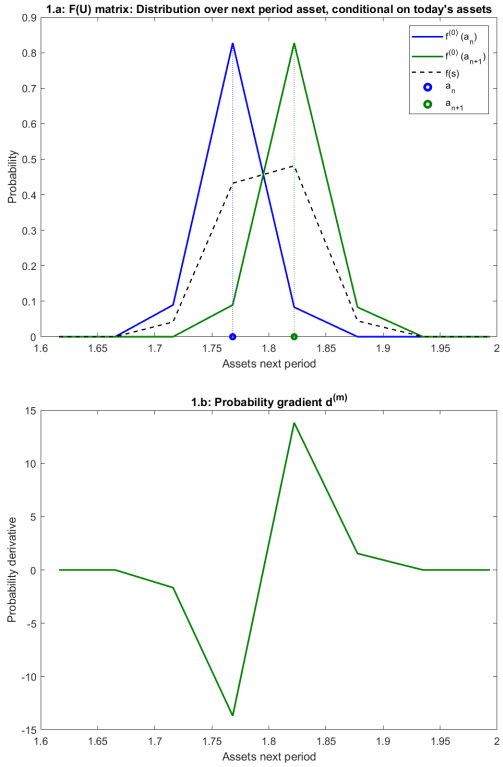
where we have defined the drift, m, and a forward difference quotient $\mathbf{d}_m$ at $a = a_n$, which captures how the distribution $\mathbf{f}(s)$ changes locally as we vary savings between to adjacent grid points, i.e. for $s \in [a_n, a_{n+1}]$.[11] $\mathbf{d}_m(a_n)$ is visualized in the Figure 1.b, where we can see that increasing savings lowers the probability of transitioning to the grid points with lower assets, but increases the probability of transitioning to grid points with higher levels of assets. The first summand on the right-hand side of (7), $\mathbf{f}^{(0)}$, is the expansion point (or what we will call the *reference distribution)*; the second summand is the first-order term of a Taylor expansion mentioned before. One conditional distribution $f(s)$ approximated in this way is visualized in the Figure 1.a.

Sketch of proposed algorithm. We now sketch our algorithm, using consumption c rather than s as a choice variable (the mapping between to two is immediate from the budget constraint). We first define a drift function in terms of state and control, $m(a; c) =$

[10] This is equivalent to the *upwind principle* for first derivatives in solution schemes in continuous-time MCA algorithms, or indeed any solution algorithm for partial differential equations.

[11] We also construct a backward quotient for negative drifts, using an analogous formula for negative drifts, $s < a_n$. We omit this and only present the forward case for brevity. It is important to note here that the use of the backward and forward quotients ensures that we obtain proper probability distributions $\mathbf{f}(s)$ for any $s \in [a_{n-1}, a_{n+1}]$, i.e. all vector elements are non-negative and sum to one: This is so due to (6): the approximating distribution is constructed as a convex combination of two proper distributions.

Figure 1: Obtaining the mixed distribution $f(s)$



$(a - c + y)r - c + y$. Next, we create a matrix $\mathbf{D}_m = [\mathbf{d}_m(a_1); \mathbf{d}_m(a_2), \dots]$, collecting the gradients.[12]. Finally, we collect the vectors $\mathbf{f}^T$ to a transition matrix on our grid, given a vector $\mathbf{c}$ of consumption choices:

$$\mathbf{F}(\mathbf{c}) = \mathbf{F}^{(0)} + m(\mathbf{a}; \mathbf{c}) \circ \mathbf{D}_m, \tag{8}$$

where $\mathbf{y} \circ \mathbf{Z} \equiv [y_i Z_{ij}]_{i,j}$ denotes element-by-element multiplication in the sense that each element y_i multiplies the entire ith row of matrix $\mathbf{Z}$. The approximate Bellman Equation that characterizes the $N - by - 1$ value vector $\mathbf{w}$ in the optimum is

$$\mathbf{w} = \beta \mathbf{F}^{(0)} \mathbf{w} + \max_{\mathbf{c}} \Big\{ \underbrace{u(\mathbf{c}) + \beta m(\mathbf{a}, \mathbf{c}) \circ \mathbf{D}_m \mathbf{w}}_{= H(a; c; \cdot)} \Big\}, \tag{9}$$

[12] Again, we restrict attention to positive drifts here, but backward quotients are used for negative drifts.

where $\circ$ denotes element-by-element multiplication of two vectors. Here, we may call $H(\cdot)$ a *pseudo Hamiltonian*.[13] As in the corresponding continuous-time problem, $H(\cdot)$ captures the dynamic effects of savings through the term $\mathbf{D}_m\mathbf{w}$, which captures the marginal value of wealth, just as the value-function derivative does in continuous time. This leads to a very simple FOC for consumption that can be solved in closed form, a simplification that leads to major speed gains in all our examples.

Finally, we simply iterate on value function given an initial guess $\mathbf{w}_0$, applying an acceleration step (also called modified policy function iteration). The algorithm is as follows:

1. Make an initial guess for the value function $\mathbf{w}_0$ and consumption $\mathbf{c}_0$. Compute the matrices $\mathbf{F}^{(0)}$ (transition probabilities without drift, i.e. for $m(a;c) = 0$), $\mathbf{D}_m$ (approximated derivatives of the transition probabilities with respect to the drift $m(a;c)$). Set the iteration counter to $n = 0$.

2. Set $n = n + 1$. Obtain $\mathbf{c}_n^*$ by evaluating the first-order condition given the approximation
$$\mathbf{c}_n^* = (\beta \underbrace{(1+r)}_{=-m_c(a;c)} \mathbf{D}_m\mathbf{w}_{n-1})^{-\frac{1}{\gamma}} \tag{10}$$

3. Given the calculated $\mathbf{c}_n^*$, update the drift $m(\mathbf{a};\mathbf{c}_n^*)$, the transition matrix $\mathbf{F}(\mathbf{c}_n^*)$ and the value function $\mathbf{w}_n = u(\mathbf{c}_n^*) + \beta\mathbf{F}(\mathbf{c}_n^*)\mathbf{w}_{n-1}$.

4. Use the acceleration step k times with the given policy function $\mathbf{c}_n^*$: $\mathbf{w}_n = u(\mathbf{c}_n^*) + \beta\mathbf{F}(\mathbf{c}_n^*)\mathbf{w}_n$

5. Compare the computed $\mathbf{c}_n^*$ with $\mathbf{c}_{n-1}^*$. If the norm $||\mathbf{c}_n^* - \mathbf{c}_{n-1}^*||$ is sufficiently small, end the algorithm. If not, set $n = n + 1$ and go to step 2.

[13] To see why, consider the corresponding continuous-time HJB with Hamiltonian H_{ct}:
$$\rho V(a) = \max_c \{ \underbrace{u(c) + (ra + y - c)V'(a)}_{=H_{ct}(a;c;V'(a))} \}.$$

2.3 Comparison: Consumption-savings

Table 1: Parametrization

Parameter	Symbol	Value
Risk aversion / IES	γ	2
Discount factor	β	0.96
Mean asset return	r	0.02
Return variance	σ	0.03
Income	y	0.2
Points in Tauchen approximation	p	3
Number of "speed-up" iterations	k	100
Number of grid-points	N	350

We calibrate the model at the yearly frequency, thus setting a standard discount factor of $\beta = 0.96$. This is a rather long time-period, which does not work in favour of the proposed method. Despite this, the algorithm produces a numerical result comparable to the other methods. As discussed earlier, shortening the time-period reduces the discrepancy in the results, as the Markov approximation becomes more accurate, and the discrete-time problem gets closer to the continuous-time problem.

Table 2: Comparison: Consumption-savings problem

Method	Run-time[b]	Solution in "linear" region[a] Consumption rate	# Iterations[c]	Euler Equation Errors [d]
Standard (exogenous) grid	0.2462	2.92%	12	6.84e-07
Endogenous grid	0.1195	3.00%	451	6.86e-07
Markov approach	0.0162	3.01%	17	6.07e-07

[a]Expressed as a percentage of assets. The policy function is evaluated when the assets are equal to 1500 in every algorithm, a region where the policy function is practically linear.

[b]Reported values are in seconds. The code is written in MATLAB. All simulations are executed on a personal computer using Windows 7 (64-bit) operating system, with Intel i7-8700 Central Processing Unit (6 cores and 12 threads), clocked at 3.19 GHz.

[c]Acceleration step is used in all algorithms except the endogenous grid method, (as it slows down the computation when using that method). Acceleration step performs $k = 100$ iterations, so the overall number of iterations is obtained by multiplying the reported number by k.

[d]The reported value is the maximum error computed in the asset interval $[1100, 1900]$, based on the computed 1000 points. Since there is no straightforward way to compute the Euler equation errors in between the grid-points for the Markov-approximation, they are computed using the Tauchen discretization of the law of motion with 5 possible states.

We see that the proposed Markov approach results in speed-gains even in relatively simple model with only one state and control variable. The policy functions do show small discrepancy across methods, the distance becoming smaller if the time period t is shortened (i.e. setting smaller discount and interest rates). Therefore, the accuracy reported here is rather conservative, since the chosen discount factor and interest rate are consistent with annual parametrisation. Still, we can see that even with an annual parametrization, the proposed method produces a reasonably close results. Comparing the Euler equation errors in the linear region (far from the borrowing constraint[14]), we can see that the proposed method has smaller residuals that benchmark methods.

[14]Figure C.1 in Appendix C shows the policy functions close to the borrowing constraint. We can see that there are some discrepancies in that region, but they do not appear to be large.

3 Portfolio choice problem

We now add a second risky asset to the problem from the previous section, thus turning the problem into one of portfolio choice. This serves to illustrate how our approach generalizes when both the mean and variance moments are used in the approximation (the default case), showcasing how the speed gains increase over traditional methods increase as we add a second moment (but keep only one state dimension).

3.1 Setting

Consider again a saver with the CRRA preferences given in (1). We now set flow income to zero, i.e. $y = 0$, in order to obtain a homogeneous setting for which a closed-form solution is available, at least in the continuous-time variant of the model. At each t, the saver first takes consumption $c_t \in [0, a_t]$ out of her wealth and then divides the remaining savings into a safe and a risky asset. The law of motion for wealth is

$$a_{t+1} = (a_t - c)(1 + x_t(\tilde{r}_{t+1} - r_f) + r_f), \tag{11}$$

where a_0 is given and x_t is the portfolio share of the risky asset. $\tilde{r}$ is distributed normally with variance σ and mean r, while r_f is the return on the safe asset with no variance. Therefore, in this problem, there are two control variables: x and c.

3.2 Proposed Markov approach

We first describe three versions of our Markov-chain approach. The benchmark algorithms we compare our approach to are described in the appendix.

To avoid negative probabilities and assure the stability the return of the safe asset r_f, which in the original problem has 0 variance is assumed to have a very small variance $\sigma_f = 0.004$. Since, we are approximating the perfectly safe asset by an asset with very small variance, the net returns on the risky and safe asset, $\tilde{r_r}$ and $\tilde{r_s}$ are joint-normally distributed with mean $\mu = \begin{pmatrix} r_r \\ r_s \end{pmatrix}$ and variance $\Sigma = \begin{pmatrix} \sigma_r^2 & \rho\sigma_r\sigma_s \\ \rho\sigma_r\sigma_s & \sigma_s^2 \end{pmatrix}$.

Table 1: Parameterization of portfolio problem

Parameter	Symbol	Value
Risk aversion / IES	γ	2
Discount factor	β	0.96
Mean risky asset return	r_r	0.022
Safe asset return	r_f	0.0195
Risky return variance	σ	0.03
Covariance coefficient	ρ	0
Points in Tauchen approximation	p	$9, 7^a$
Number of "speed-up" iterations	k	100
Number of grid-points	N	400

[a] 9 for in the exogenous grid, and 7 in endogenous grid method, which is the smallest odd number to achieve stability in this particular application.

The idea of our algorithm is now as follows: We approximate the conditional distribution of assets given a choice (c, x) by approximating the distribution as linear combinations of certain benchmark moments. We first write down functions that give us the mean and variance of assets conditional on controls (c, x) and current assets a. The drift function is

$$m(a; c, x) = (a - c)(r_f + x(r_r - r_f)) - c,$$

the variance function is

$$v(a; c, x) = (a - c)^2 \left[(1 - x)^2\sigma_f^2 + 2x(1 - x)\sigma_{fr} + x^2\sigma_r^2\right],$$

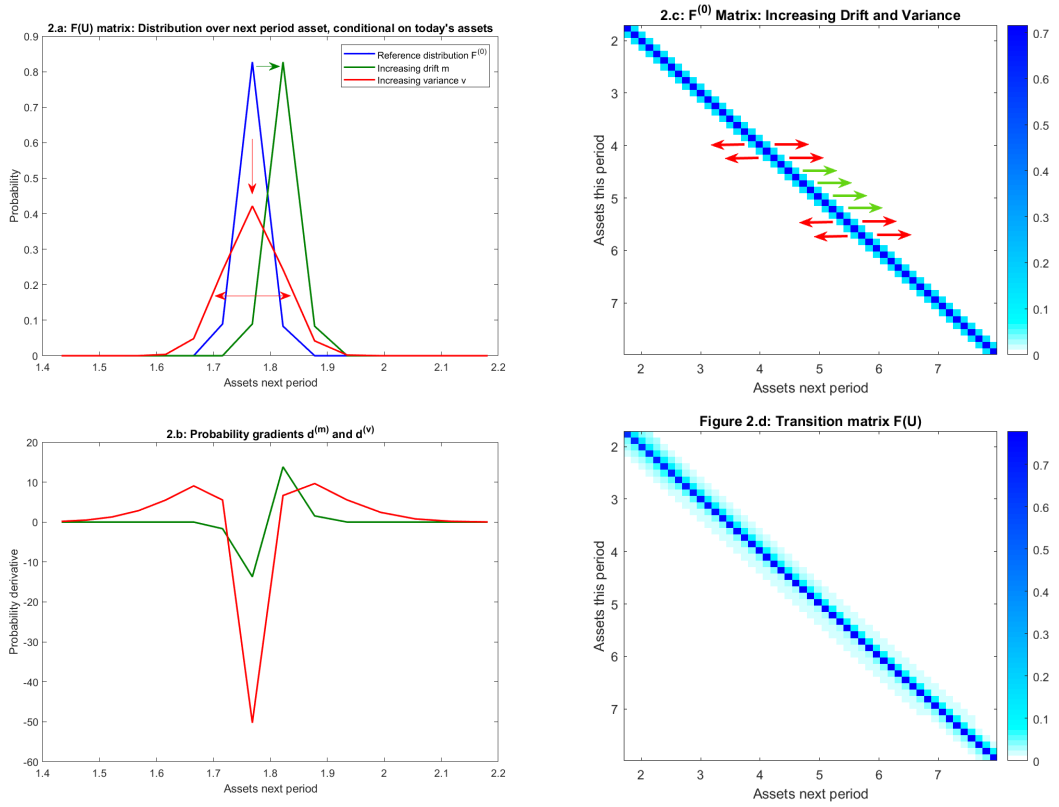
where $\sigma_{fr} = \rho\sigma_r\sigma_f$ is the covariance between the two assets.

Our **linear moment approximation step** proceeds then as in the simple consumption-savings model, but now expanding with respect to two moments: the mean and the variance.[15] Again, we first choose a reference point, in this case the one characterized $\bar{m} = 0$

[15] We show in the appendix that choosing the variance as the parameter governing the dispersion gives us first-order accurate approximating variances. The same is *not* true if we chose the standard deviation instead of the variance, for example. This is in line with the results from (Kushner et al. 2001) that show that mean and variance of approximating Markov chains have to be accurate (at least to a first order) to guarantee good convergence properties.

(no drift in assets) and $\bar{v} = \sigma_f^2$ (portfolio only in safe asset), from which we obtain the reference distribution $\mathbf{f}^{(0)}(a_n)$ at grid point a_n. Figure 2.a shows this distribution in blue. To determine how the distribution changes as m and v increase, we then calculate distributions i) varying only the drift m (but fixing the variance $\bar{v}$) to obtain the gradient $\mathbf{d}_m(a_n)$ and ii) varying only the variance v (but fixing the drift $\bar{m}$) to obtain $\mathbf{d}_v(a_n)$. This procedure is depicted in Figure 2. Sub-figures 2.a and 2.b echo Figure 1, but in addition to the drift, now we have an additional moment in the approximation: v. We see that the derivative $\mathbf{d}_v$ puts more weight on the tails of the distribution, removing it from the center.

Figure 2: Increasing the drift $m(a;c)$ and and variance $v(a;c,x)$ in the matrix $F(U)$ as compared with the $F^{(0)}$ matrix



Collecting again for all asset grid points to matrices, we then have the first-order ap-

proximation[16]

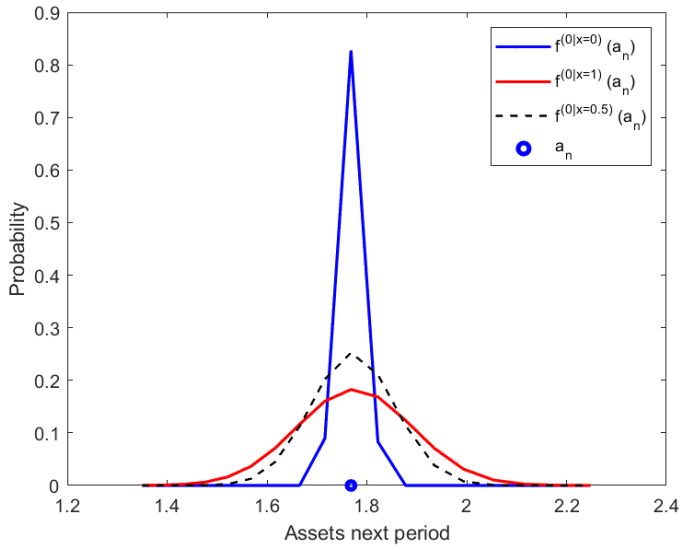
$$\mathbf{F}(\mathbf{U}) \simeq \mathbf{F}^{(0)} + m(a; c, x)\mathbf{D}_m + v(a; c, x)\mathbf{D}_v,$$

where $\mathbf{U} = [\mathbf{c}, \mathbf{x}]$ is the N-by-2 matrix of controls chosen. Again, we elements below a tolerance value to zero. Figure 2.c visualises the sparse matrix $F^{(0)}$, which contains information of a reference distribution of transition probabilities when $s = a$. It answers the question: What are the probabilities of moving to grid-points with certain amount of assets next period (horizontal axis), conditional on the starting grid-point this period (vertical axis) and saving $s = a$. Notice that it is a very sparse matrix, which our algorithm exploits. Most mass is around the diagonal, as there is a high probability to remain at exactly the same grid point next period. If the agent decides to save more than a (increasing the drift $m(a; c, x)$), then she shifts the distribution rightwards (green arrows in the figure), i.e. increases the probabilities of moving to a grid point with higher assets, and reduces the probabilities of moving to grid points with lower level of assets. Furthermore, the agent can also increase the variance of the distribution $v(a; c, x)$ by choosing a riskier portfolio with higher x. This stretches the distribution (red arrows), increasing the mass at the tails of the distribution, and decreasing the probability to stay at exactly the same grid-point. Figure 2.d visualises the transition matrix $F(U)$ under the optimal policies, which imply positive savings and some risk-taking. Notice that the distribution is still close to the diagonal, but shifted rightwards, as the agent in this region saves, $s > a$, and has positive share of a higher-yielding risky asset in their portfolio x. Furthermore, the distribution also has thicker tails, again because of the positive share of risky assets, $x > 0$.

In the previous section, Figure 1.a showed how the reference distributions mix with the changing drift and produce the mixed distribution of transitional probabilities, conditional on the chosen drift. Since we are approximating with the first two moments in the portfolio choice problem, we mix the distributions with different variance. The principle is shown in Figure 3. The reference distribution is blue, while the distribution assuming maximum

[16] We explain in the appendix who it can be ensured that we obtain a proper probability distribution in this approximation. Intuitively, this can be guaranteed as long as we construct a convex combination of three distributions (reference, high mean, high variance).

Figure 3: Variance mixing



variance ($x = 1$) is shown in red color. By choosing higher x (and therefore higher variance v), the agent moves away from the (reference) blue distribution and closer to the red distribution.

Given the linear moment approximation, the **approximate Bellman Equation** pinning down the value function $\mathbf{w} = W(\mathbf{a})$ on the grid is then

$$\mathbf{w} = \beta \mathbf{F}^{(0)} \mathbf{w} + \max_{c \geq 0, x \in [0,1]} \Big\{ \underbrace{u(c) + \beta \big[m(a;c,x) \mathbf{D}_m \mathbf{w} + v(a;c,x) \mathbf{D}_v \mathbf{w} \big]}_{\equiv H(a;c,x;\cdot)} \Big\}. \tag{12}$$

Again —and this is no coincidence—the *pseudo Hamiltonian* $H(\cdot)$ takes almost exactly the same form as in the continuous-time version of this portfolio problem: $\mathbf{D}_m \mathbf{w}$ is equivalent to the first derivative of the value function, $V'(a)$, in the continuous-time version, capturing the marginal value of wealth. $\mathbf{D}_v \mathbf{w}$ is closely related to the second derivative, $V''(a)$, which is negative and captures risk aversion. We will shortly see how this leads to an intuitive first-order condition for the portfolio choice x, capturing a mean-variance trade-off as is common in finance.

From the Bellman Equation (12), at grid point a we obtain the consumption **FOC**

and portfolio-share FOC

$$\underbrace{m_x(a;c,x)}_{=(a-c)(r_r-r_f)} D_m W + v_x(x) D_v W = 0,$$

which shows the mean-variance trade-off faced by the saver. Here, we have denoted by $D_m W$ and $D_v W$ the elements in the vectors $\mathbf{D}_m \mathbf{w}$ and $\mathbf{D}_v \mathbf{w}$ that correspond to grid point a. Also, the derivatives of drift and variance with respect to the controls are given by

$$m_c(a;c,x) = -(1 + r_f + x(r_r - r_f)), \tag{13}$$

$$m_x(a;c,x) = (a-c)(r_r - r_f), \tag{14}$$

$$v_c(a;c,x) = 2(a-c)\left[-\sigma_f^2(1-x)^2 - x(1-x)\sigma_{fr} - \sigma_r^2 x^2\right], \tag{15}$$

$$v_x(a;c,x) = \left[-2(1-x)(a-c)^2\sigma_f^2 + (1-2x)2(a-c)^2\sigma_{fr} + 2x(a-c)^2\sigma_r^2\right]. \tag{16}$$

Invoking the CRRA utility function $u(c) = \frac{c^{1-\gamma}}{1-\gamma}$, we obtain a system of two non-linear equations in two unknowns c and x:

$$c^{-\gamma} - (1 + r_f + x(r_r - r_f))\beta D_m W + \beta 2(a-c)\left[-\sigma_f^2(1-x)^2 - x(1-x)\sigma_{fr} - \sigma_r^2 x^2\right] D_v W = 0 \tag{17}$$

$$(a-c)(r_r - r_f) D_m W + \left[-2(1-x)(a-c)^2\sigma_f^2 + (1-2x)2(a-c)^2\sigma_{fr} + 2x(a-c)^2\sigma_r^2\right] D_v W = 0 \tag{18}$$

This system has no closed-form solution. We thus consider different versions of our MCA algorithm: In the first, we find a precise solution to the system of FOCs by root-finding. In the second and third, we use an additional approximation step (**law-of-motion approximation**) to simplify the system of FOCs. Again, the idea is to use an approximation around a reasonable reference point and is inspired by the way that continuous-time algorithms drop lower-order terms.

3.3 Different versions of value function iteration algorithm

3.3.1 "Fully" non-linear iteration

This is a version most similar to the classic value function iteration. The procedure is as follows:

1. Guess starting values $\mathbf{w}_0$ and $\mathbf{c}_0, \mathbf{x}_0$. Compute the matrices $\mathbf{F}^{(0)}$ (transition probabilities at zero drift, i.e. $m(a, x, c) = 0$), $\mathbf{D}_m$ (approximated derivatives of the transition probabilities with respect to the drift m), and $\mathbf{D}_v$ (approximated derivatives of the transition probabilities with respect to the variance v).

2. Set the iteration counter to $n = n+1$. Given $\mathbf{w}_n$, solve the system of equations given by equations 17 and 18 for $\mathbf{c}_n$ and $\mathbf{x}_n$ using a double (nested) root-finding routine.

3. Given the calculated policies , update $\mathbf{w}_n$ using the Bellman Eq. (12) for k times ($k > 1$: acceleration step).

4. Compare the computed c_n, x_n with c_{n-1}, x_{n-1}. If the norm is sufficiently small, end the algorithm. If not, set $n = n + 1$ and go to step 2.

The downside of this algorithm is that we need to solve a non-linear system of equations (equations 17 and 18) in each iteration. However, with some additional simplifying assumptions (inspired by the continuous-time approximation) this can be avoided. Namely, one can approximate the true solution by cancelling out some cross terms which converge to zero as we decrease the time period. We will now describe how this works (at a tolerable loss of precision) in the two options described below.

3.3.2 Cross-terms eliminated

In our first modification, we eliminate some cross terms by eliminating second-order terms in the law of motion, dropping lower terms as this would occur in the continuous-time limit. To see which terms are of second order, we write out flow controls rates, i.e. we write $c = C\Delta t$, where C is a consumption rate. We note that both x and a are stocks, thus no modification occurs here. The law of motion is then

$$a_{t+\Delta t} = (a_t - C\Delta t)[1 + r\Delta t + x(\tilde{r} - r)\Delta t].$$

Approximating to a first order, we drop terms in Δt^2 and replace again $C\Delta t$ by c, arriving at the law of motion

$$a_{t+1} = a_t(1 + r + x(\tilde{r} - r)) - c_t.$$

We can also conceive this approximate law of motion as an "altered timing protocol": It says that returns are gained on beginning-of-period assets a_t and only after that consumption c_t is taken out of the wealth stock.

Thus, we now have drift $m(a, x, c) = a(r_f + x(r_r - r_f)) - c$ and variance $v(a, x) = a^2\left[(1-x)^2\sigma_f^2 + 2x(1-x)\sigma_{fr} + x^2\sigma_r^2\right]$, the derivatives in controls being:

$$m_c(a; c, x) = -1$$

$$m_x(a; c, x) = a(r_r - r_f)$$

$$v_c(a; c, x) = 0$$

$$v_x(a; c, x) = \left[-2(1-x)a^2\sigma_f^2 + (1-2x)2a^2\sigma_{fr} + 2xa^2\sigma_r^2\right]$$

If we take FOCs now, we get:

$$c^{-\gamma} - \beta D_m W = 0 \tag{19}$$

$$a(r_r - r_f)D_m W + a^2\left[-2(1-x)\sigma_f^2 + (1-2x)2\sigma_{fr} + 2x\sigma_r^2\right]D_v W = 0 \tag{20}$$

which is convenient, because we can get solutions for c and x in closed form and not dependent on each-other directly. This means that now in the step 2, we don't have to solve the system of non-linear equations, which is computationally expensive. Instead, we can simply evaluate two equations to obtain optimal c and x:

$$c^* = (\beta D_m W)^{\frac{-1}{\gamma}}$$

$$x^* = \frac{(r_r - r_f)D_m W + (-\sigma_f^2 + \sigma_{fr})2aD_v W}{(\sigma_f^2 - 2\sigma_{fr} + \sigma_r^2)2aD_v W}$$

which significantly reduces the computational time.

3.3.3 Cross-terms "exogenous" / constant

The simplification in the previous section greatly reduced computation time by eliminating the "cross-terms" where optimal choice of c depended on x and vice-versa. However, a by-product is an approximation error. This section develops an approach which reduces this approximation error, but still preserves the computational gain from the previous section.

Now, instead of completely eliminating cross-terms, we are approximating them by a best guess. In other words, $(a - c)$ in the drift and variance derivatives will now be approximated by $(a - \bar{c})$, where $\bar{c}$ will be treated as an exogenous constant when taking the FOCs. For $\bar{c}$, we use the solution from c from the precious iteration $\bar{c}_n = c^*_{n-1}$ in practice. In general, the modified equations can be worked out approximating $c = \bar{c} + \Delta c$ and $x = \bar{x} + \Delta x$ in the FOCs, taking a Taylor expansion of the FOCs in $(\Delta c, \Delta x)$ around $(0, 0)$ and dropping the lowest-order terms.

Following this method, drift and variance derivatives become:

$$
\begin{aligned}
m_c(a; c, x) &= -(1 + r_f + x(r_r - r_f)) \\
m_x(a; c, x) &= (a - \bar{c})(r_r - r_f) \\
v_c(a; c, x) &= 2(a - \bar{c}) \left[-\sigma_f^2(1 - x)^2 - x(1 - x)\sigma_{fr} - \sigma_r^2 x^2\right] \\
v_x(a; c, x) &= \left[-2(1 - x)(a - \bar{c})^2\sigma_f^2 + (1 - 2x)2(a - \bar{c})^2\sigma_{fr} + 2x(a - \bar{c})^2\sigma_r^2\right]
\end{aligned}
$$

Now, it is possible to compute the optimal x^*, and then given the optimal x^*, compute the optimal c^*. This way, we still avoid solving a system of equations, but the approximating error is smaller, since $|c - \bar{c}| < |c - 0|$.

FOCs now give:

$$
x^* = \frac{(r_r - r_f)D_m W + (-\sigma_f^2 + \sigma_{fr})2(a - \bar{c})D_v W}{(\sigma_f^2 - 2\sigma_{fr} + \sigma_r^2)2(a - \bar{c})D_v W} \tag{21}
$$

$$c^* = \left\{\beta(1 + r_f + x(r_r - r_f))D_m W + \beta 2(a - \bar{c})\left[-\sigma_f^2(1-x)^2 - x(1-x)\sigma_{fr} - \sigma_r^2 x^2\right] D_v W\right\}^{\frac{-1}{\gamma}} \tag{22}$$

Here, we replace the step 2 of the above algorithm with:

- Obtain x^* by evaluating (21) and then c^* by evaluating (22). Given the calculated c_n^* and x_n^*, update V ($D_m W$ and $D_v W$). Set $\bar{c} = c^*$.

In addition we need to approximate the term $(a - c)$ with $(a - \bar{c})$ in step 1. The rest of the algorithm is as described in the section 3.3.1.

3.4 Comparison: Portfolio choice problem

Table 4: Portfolio choice (Merton) problem

Method	Run-time[b]	Solution in "linear" region[a] Consumption	Portfolio	# Iterations[c]	Euler Eq'n Errors [d]
Benchmark methods:					
Standard (exogenous) grid	12.6886	2.94%	27%	14	6.45e-07
Endogenous grid	5.0449	3.04%	27.9%	484	1.367e-06
Markov approach:					
Exact	0.7154	2.95%	30.1%	8	5.33e-07
Modified timing	0.0106	3.01%	29.1%	8	8.14e-07
Taylor approximation	0.0146	2.95%	29.9%	9	5.34e-07
Cont. time closed-form solution		2.99%	29.5%		

[a]Consumption expressed as a percentage of assets; portfolio is risky share of savings. The policy function is evaluated when the assets are equal to 1500 in every algorithm, a region where the policy function is practically linear.

[b]Reported values are in seconds. The code is written in MATLAB. All simulations are executed on a personal computer using Windows 7 (64-bit) operating system, with Intel i7-8700 Central Processing Unit (6 cores and 12 threads), clocked at 3.19 GHz.

[c]Acceleration step is used in all algorithms except the endogenous grid method, (as it slows down the computation when using that method). Acceleration step performs $k = 100$ iterations, so the overall number of iterations is obtained by multiplying the reported number by k.

[d]The reported value is the maximum error computed in the asset interval $[1000, 2000]$, based on the computed 1000 points. Since there is no straightforward way to compute the Euler equation errors in between the grid-points for the Markov-approximation, they are computed using the Tauchen discretization of the law of motion with 9 possible states.

Table 4 summarizes the most important results in Section 3. First, we can see that the policy functions of the proposed methods are quite close to the ones obtained by the standard benchmark methods. This is despite relatively large discount factor ($\beta = 0.96$), consistent again with an annual calibration. Second, our solutions are generally closer to the closed-form continuous-time solution. Third, measured by Euler equation residuals, the proposed methods are not less accurate than the benchmark models. Fourth, the Markov approach algorithms are significantly faster than the benchmark models, the improvement ranging from a factor of 8 to a factor of 300, depending on which implementation is chosen. Fifth, the Taylor approximation implementation of the Markov approach, which signifi-

cantly reduces the computational time, still produces relatively small Euler equation errors, whereas the modified timing implementation is somewhat less precise.

Table 5: Varying the number of grid-points N

	Runtime (seconds)		
Method	$N = 150$	$N = 300$	$N = 450$
Benchmark methods:			
Standard (exogenous) grid	8.988	10.273	14.445
Endogenous grid	2.945	3.934	5.765
Markov approach:			
Exact	0.250	0.553	0.840
Modified timing	0.008	0.011	0.015
Taylor approximation	0.009	0.013	0.016

4 A game-theoretic application: altruistically linked savers

We now present an example that shows that the approach is also feasible, and in fact advantageous, when i) the dimensionality of the state space is high and ii) when multiple decision makers are present.

4.1 Setting

There are two infinitely-lived agents, a parent (indexed by p) and a child (indexed by k for "kid"). Both have access to savings at a fixed gross interest rate $R > 0$ and cannot borrow, that is we require $a^p \geq 0$ and $a^k \geq 0$, where a^i are agent i's assets. In each period $t = 0, 1, \ldots$, the parent first chooses consumption c_t^p and a transfer $g_t \geq 0$ (where g is for gift). After observing the transfer, the child then chooses her consumption c_t^k. Savings s_t^i for $i \in \{p, k\}$ are then residually determined from the budget constraints, which are

$$c_t^p + g_t + s_t^p = a_t^p + y_t^p, \tag{23}$$

$$c_t^k + s_t^k = a_t^k + g_t + y_t^k. \tag{24}$$

Here, y_t^i is agent i's current income. y_t^i follows a Markov chain with a set of states $\{\hat{y}^i\}_{i=1}^{N_{yi}}$ and transition probabilities $\pi_{yi}(\hat{y}^i|\hat{y}^j)$ from state j to i. The return R^i on savings s^i for both $i \in \{p, k\}$ follows a log-normal distribution with parameters (r, σ), that is,

$$\ln R^i = r + \sigma\epsilon^i \qquad \text{, where } \epsilon^i \sim \mathcal{N}(0, 1). \tag{25}$$

Here, r governs the mean and σ the dispersion of returns. For simplicity, we assume that the returns are independent across agents and across time; they are also assumed to be independent from the income shocks. The law of motion for assets is $a_{t+1}^i = R_{t+1}^i s_t^i$, from which it follows that

$$\ln a^{i'} = \ln s^i + r + \sigma\epsilon^{i'} \qquad \Rightarrow \qquad \ln a^{i'} \sim \mathcal{N}(s^i + r, \sigma), \tag{26}$$

where we denote one-period-ahead variables with primes. This pins down the distribution $f^{(i)}(a^{i'}|s^i)$ for agent i's future assets given her savings choice as a log-normal distribution

as given in (26). Agents' preferences over consumption sequences are

$$V_0^p = \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \big[u(c_t^p) + \alpha u(c_t^k)\big], \tag{27}$$

$$V_0^k = \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t u(c_t^k), \tag{28}$$

where $\beta \in (0,1)$ is a discount factor common to both agents, $\alpha > 0$ captures the parent's altruism towards the child and where $u(\cdot)$ is a utility functional satisfying the standard assumptions $u' > 0$ and $u'' < 0$ as well as the Inada condition $\lim_{c\to 0} u'(c) = \infty$.

4.2 Agents' problems and Bellman Equations

As described in Barczyk & Kredler (2014), we transform this problem to a game in which agents choose actions simultaneously each period. We let the child announce *desired consumption* $\tilde{c}_t^k > 0$, which need not satisfy the budget constraint — note that transfers by the parent can make consumption in excess of the child's own resources possible. Simultaneously to the child's choice $\tilde{c}_t^k$, the parent chooses c_t^p and g_t. We then determine *realized* consumption (over which preferences are defined), c_t^k, from the following equations:

$$c_t^k = c^*(\tilde{c}_t^k, \tilde{a}_t^k) \equiv \min\{\tilde{c}_t^k, \tilde{a}_t^k\}, \tag{29}$$

$$\text{where } \tilde{a}_t^k = a_t^k + y_t^k + g_t. \tag{30}$$

Here, $\tilde{a}_t^k$, as defined by (30), is the child's cash-on-hand at t after having received the parent's transfer. The function $c^*(\cdot)$, as defined by (29), gives realized consumption and says that the child consumes up to her desired level as long as parent transfers allows this (and saves anything that exceeds desired consumption, $\tilde{c}_t^k$).

We now look for a Markov-Perfect Equilibrium (MPE) of the dynamic game. The game's payoff-relevant state is given by the vector $\mathbf{x} \equiv (a^p, a^k, y^p, y^k) \in \mathbb{R}^4$. The system

of Bellman Equations that characterizes a MPE is

$$W^p(\mathbf{x}) = \max_{c^p, g} \Big\{ u(c^p) + \alpha u\big(c^*(\tilde{c}^k, \tilde{a}^k)\big) + \beta \mathbb{E}_{\epsilon^{p\prime}, \epsilon^{k\prime}, y^{p\prime}, y^{k\prime}} W^p(a^{p\prime}, a^{k\prime}, y^{p\prime}, y^{k\prime}) \Big\}, \quad (31)$$

$$W^k(\mathbf{x}) = \max_{\tilde{c}^k} \Big\{ u\big(c^*(\tilde{c}^k, \tilde{a}^k)\big) + \beta \mathbb{E}_{\epsilon^{p\prime}, \epsilon^{k\prime}, y^{p\prime}, y^{k\prime}} W^k(a^{p\prime}, a^{k\prime}, y^{p\prime}, y^{k\prime}) \Big\} \quad (32)$$

s.t. (30),

$$\ln a^{p\prime} = \ln(a^p + y^p - c^p - g) + r + \sigma \epsilon^{p\prime}, \quad (33)$$

$$\ln a^{k\prime} = \ln(\tilde{a}^k - c^*(\tilde{c}^k, \tilde{a}^k)) + r + \sigma \epsilon^{k\prime}, \quad (34)$$

where the last two equations follow from the budget constraints and the the law of motion for assets. The first two equations are the usual Bellman Equations, noticing here that the agents' choices also have effects on the other player. We now solve this system using our algorithm, guessing that optimal transfers satisfy the *transfers-when-constrained* property: Parent gifts only flow when the child has zero assets and the child always consumes up any gift from the parent.

Independent FOCs. As in the basic consumption-savings model from Section 2, players' choices affect only the mean of their own asset distribution, but not the variance parameter nor the other agent's assets. When approximating the conditional distribution to a first order in its moments, it turns out that interactions between agents' contemporaneous choices are of second order and the agent's consumption-savings choice becomes independent of what the other agent chooses at the same moment, thus simplifying computation enormously[17] It is not a coincidence that this independence is exactly what Barczyk & Kredler (2014) find in the continuous-time limit, since also there higher-order terms vanish. The economic intuition is as follows: The other player's current choice only moves

[17] ...although the other agent's future choices matter, as they show up in the continuation value. Mathematically, independence of shocks implies that the conditional density function over states $\mathbf{x}'$ is given by a product over the conditional densities in each dimension, i.e. $f(\mathbf{x}'|\mathbf{x}, s^p, s^k) = f^{(p)}(a^{p\prime}|s^p) f^{(k)}(a^{k\prime}|s^k) \ldots$, where the savings choices s^p and s^k govern the mean moments in the asset dimensions. When approximating f to a first order around reference values $(\bar{s}^p, \bar{s}^k)$, only $\bar{s}^k$ shows up in the parent's first-order condition, but not the child's $s^k - \bar{s}^k$ from the reference policy. This is because

$$\frac{\partial f(\mathbf{x}'|\cdot)}{\partial s^p} = \frac{\partial f^{(p)}(\cdot)}{\partial s^p} f^{(k)}(a^{k\prime}|\bar{s}^k) \ldots$$

the state variables by very little (since consumption is a flow variable), thus leading to only small changes in the marginal value of savings. It has to be noted that the first-order approximation in our discrete-time setting here is good under two conditions: i) the time horizon is short (which implies that the state does not stray far from its reference value) and ii) the continuation value is differentiable in the asset dimensions (which is ensured by the noise in the asset variables).

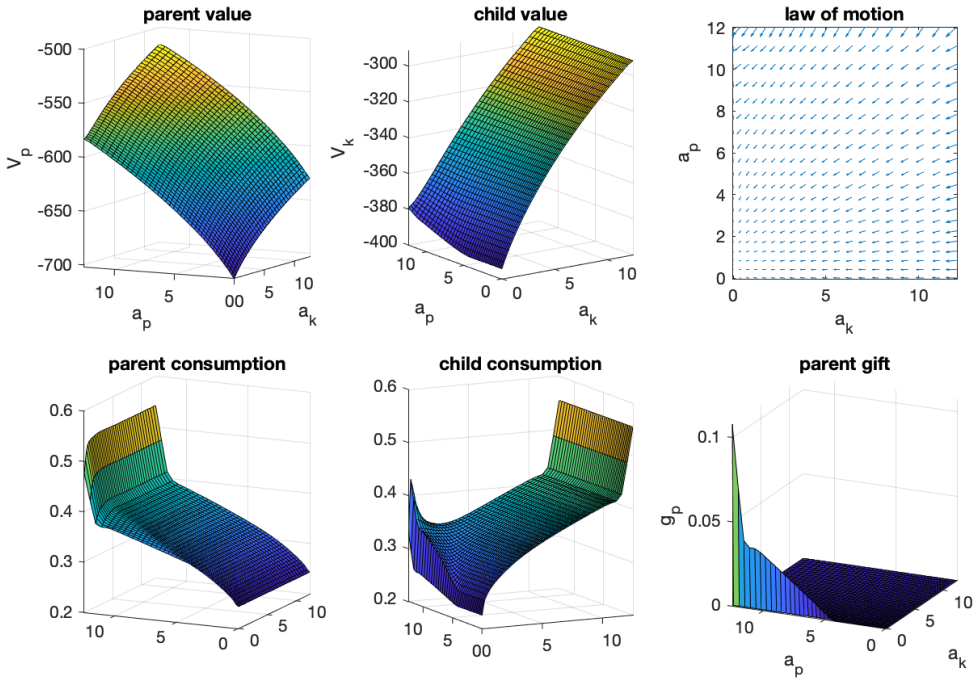
4.3 Results and comparison of algorithms

Results. Figure 4 shows the equilibrium. We notice that its properties are exactly in with Figures 2-4 in Barczyk & Kredler (2014), when adjusting for the fact that the child is not altruistic and thus no transfers flow child-to-parent and that parameter values are different. The transfer function is linearly increasing in parent wealth, transfers flowing only when the child is constrained and from some threshold level of parent wealth on. Consumption is only a function of agents' own assets in the part of the state space in which the child is relatively rich and gifts are unlikely in the near future. However, the child starts to consume more (and the parent less) than once the economy is close to the transfer region (which is the termed the "overconsumption (OC) region" by Barczyk & Kredler (2014)). Finally, we observe the characteristic discrete drop in child consumption once the child receives transfers.[18]

Gains from tensor approach. In this high-dimensional example (4D), it turns out that it is not advantageous to calculate the full transition matrix of states for the approximating Markov chain. We call this explicit approach the *Markov-Chain (MC) Kronecker* method since it requires a Kronecker product of the four transition matrices for each dimension. Figure 5 shows the time savings that occur from a different approach (*MC Tensor*) that avoids calculating this large transition matrix. We see that the time savings are very substantial and that computation time increases much slower in grid size under the MC Tensor

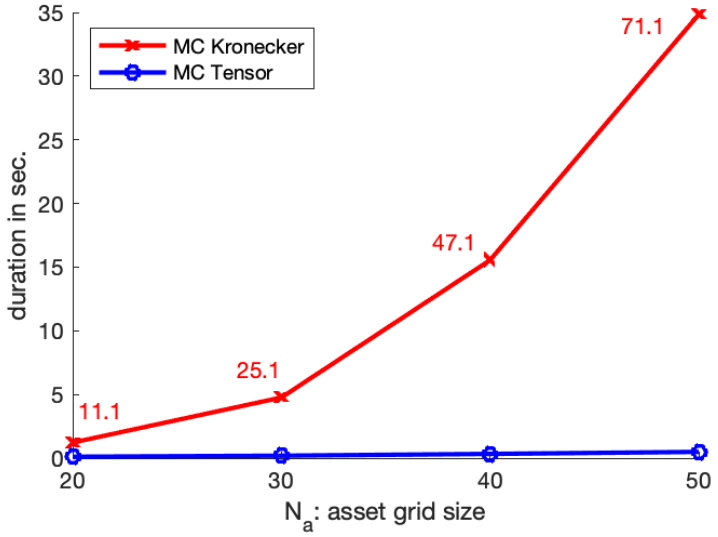
[18] Also, we observe that the transfer and consumption functions increase sharply at the upper end of the asset grids, which occurs because we reflect assets back into the grid for high shocks, thus disincentivizing savings in this region. In applications, this region should be discarded, i.e. the economy should only reach this region with negligible probability, which can be achieved by choosing the upper bound of the asset grid high enough. Unlike Barczyk & Kredler (2014) we show this region here since the computational algorithm and not the economic problem is of main interest here.

Figure 4: Equilibrium in altruism model



Parameters (1 period $\simeq$ one quarter): $\gamma = 2$, $\alpha = 0.5$, $\beta = 0.99$, $r = 0.005$, $\sigma = 0.025$, $\hat{\mathbf{y}}^p = [0.2, 0.25]$, $\hat{\mathbf{y}}^k = [0.225, 0.275]$, income transition matrix: $\mathbf{F}^{(y)} = [0.95, 0.05; 0.025, 0.975]$. Grids: $N_a = 40$, $N_y = 2$. Results shown for high parent and low child income.

Figure 5: Algorithm performance in altruism model



Duration of value-function iteration loop in altruism model (parameters as in Fig. 4), Markov-Chain (MC) Kronecker vs. MC Tensor algorithm. Red numbers next to data points show ratio between MC Kronecker and MC Tensor duration.

than under the MC Kronecker approach, thus avoiding the curse of dimensionality.

How does MC Tensor realize these large speed gains (more than 70-fold for asset grid size 50)? We give a brief intuition here for a simple 2D example that we sketch in Fig. 6; Prop. B.1 in the appendix provides a formal result that allows to estimate the number of operations necessary under each approach.[19] In the figure, the MC Kronecker approach follows each single "path" when computing a continuation value, thus computing all products in the expectation $\mathbb{E}_{x_1',x_2'}[V']$ with associated probabilities $F^{(1)}_{j_1\cdot}F^{(2)}_{j_2\cdot}$.[20] In the 2D case, this leads to a total of N_1N_2 paths; with more dimensions this number increases multiplicatively with the number of grid points. However, there is another, sequential, way of attacking the problem that is more in the spirit of dynamic programming: MC Tensor exploits the tensor structure of the problem that follows from independence of shocks.[21] We can bundle

[19] These speed savings are known in the literature, see Fackler (2019) and the references therein. We provide a proposition here to have all relevant results in one place.

[20] We focus here on the (backward) computation of continuation values, but the same gains obtain for computing any other expectation or for mapping forward distributions over time.

[21] Or, more precisely, from the fact that transition probabilities on the large state space are given by simple products of the transition probabilities in the single dimensions.

information in shorter stages and apply the shocks in the two dimensions sequentially, first calculating an intermediate value function $\mathbb{E}_{x_2'} l[V']$ and then using this to find the continuation value $V = \mathbb{E}_{x_2'}[\mathbb{E}_{x_1'}[V']]$. The number of multiplications is only $N_1 + N_2$ in this example, and it grows additively (and not multiplicatively) as we add more dimensions.

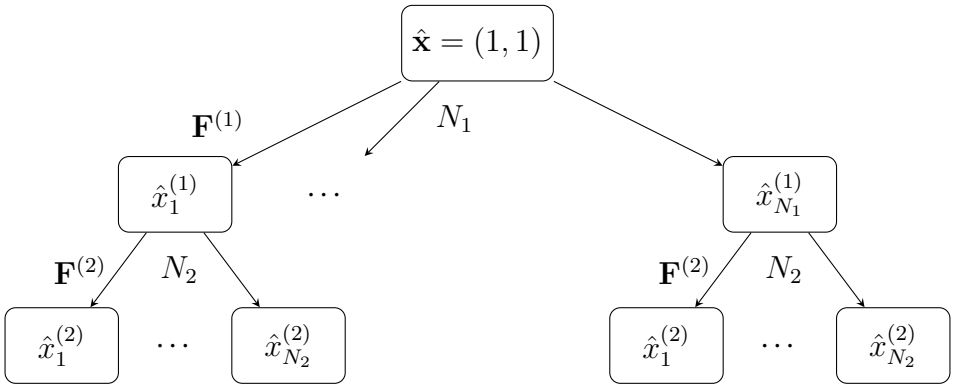


Figure 6: 2D Markov chain with independent shocks

Diagram for Markov chain with $n = 2$, starting at one fixed grid point $\hat{\mathbf{x}}$. Solid arrows: state transitions. Caption inside boxes: new state after transition. Captions left of arrows: transition matrix associated with transition. Captions between arrows: number of possible transitions in stage.

5 Conclusion

We have proposed a Markov-chain approximation method for discrete-time control problems in order to reap the computational benefits of the continuous-time algorithms. We achieve this by taking a first-order approximation of conditional distributions in their first and second moments around a reference point. We show how to apply the method to standard consumption-savings problems and portfolio choice problems. Speed gains of the proposed model are up to two orders of magnitude compared to the state-of-the-art endogenous grid method. At the same time, we observe no significant sacrifices in terms of algorithm accuracy. Measured both policy functions and Euler equation residuals, the proposed approach does not appear less accurate than the benchmark methods, even in a model parametrized to a (relatively long) annual frequency.

We show how our method avoids the curse of dimensionality and keeps computation times manageable in high-dimensional problems with independent shocks. Furthermore, our approach can substantially simplify the computation of dynamic games with a large state space, solving a discrete-time version of the altruistic savings game studied by Barczyk & Kredler (2014).

Finally, the benefits and potential drawbacks of the proposed method are discussed, and the general recipe and recommendations are provided.

References

Achdou, Y., Han, J., Lasry, J.-M., Lions, P.-L. & Moll, B. (2022), 'Income and wealth distribution in macroeconomics: A continuous-time approach', *The Review of Economic Studies* **89**(1), 45–86.

Bader, B. W. & Kolda, T. G. (2006), 'Algorithm 862: Matlab tensor classes for fast algorithm prototyping', *ACM Transactions on Mathematical Software (TOMS)* **32**(4), 635–653.

Barczyk, D. & Kredler, M. (2014), 'Altruistically motivated transfers under uncertainty', *Quantitative Economics* **5**(3), 705–749.

Carroll, C. D. (2006), 'The method of endogenous gridpoints for solving dynamic stochastic optimization problems', *Economics Letters* **91**(3), 312–320.

De Lathauwer, L., De Moor, B. & Vandewalle, J. (2000), 'A multilinear singular value decomposition', *SIAM journal on Matrix Analysis and Applications* **21**(4), 1253–1278.

Fackler, P. L. (2019), 'Algorithm 993: Efficient computation with kronecker products', *ACM Transactions on Mathematical Software (TOMS)* **45**(2), 1–9.

Heer, B. & Maußner, A. (2011), 'Value Function Iteration as a Solution Method for the Ramsey Model', *Journal of Economics and Statistics (Jahrbuecher fuer Nationaloekonomie und Statistik)* **231**(4), 494–515.
URL: *https://ideas.repec.org/a/jns/jbstat/v231y2011i4p494-515.html*

Karatzas, I., Shreve, S. E., Karatzas, I. & Shreve, S. E. (1998), *Methods of mathematical finance*, Vol. 39, Springer.

Kushner, H. J. K., Kushner, H. J., Dupuis, P. G. & Dupuis, P. (2001), *Numerical methods for stochastic control problems in continuous time*, Vol. 24, Springer Science & Business Media.

Phelan, T. & Eslami, K. (2021), 'Applications of markov chain approximation methods to optimal control problems in economics'.

Puterman, M. L. & Brumelle, S. L. (1979), 'On the convergence of policy iteration in stationary dynamic programming', *Mathematics of Operations Research* **4**(1), 60–69.

Tauchen, G. (1986), 'Finite state markov-chain approximations to univariate and vector autoregressions', *Economics letters* **20**(2), 177–181.

A The cookbook recipe

In this appendix, we explain the algorithm that we propose in general terms. We will denote vectors by bold lower-case letters (e.g. $\mathbf{a}$), matrices by bold upper-case letter ($\mathbf{A}$) and functions and scalar variable by plain lower-case letters (x, $f(\cdot)$). When indexing single elements of vectors and matrices, we drop the bold face and write a_j, $A_{i,j}$; we use brackets to collect all elements to a vector or, e.g. $\mathbf{A} = [A_{i,j}]$. To index an entire matrix column, we write $A_{\cdot j}$, where the dot indicates that the index i runs over all positions, thus returning a column vector. We denote the matrix transpose by a T-superscript ($\mathbf{A}^T$) and reserve primes ($\mathbf{x}'$) for one-period-ahead variables, following the notation that is standard in dynamic programming. We let the operator $\circ$ denote element-by-element multiplication; in the case of multiplication of a vector by a matrix, $\mathbf{a} \circ \mathbf{B}$, it is understood that all elements of jth row of $\mathbf{B}$ are multiplied by the vector element a_j. We denote objects in the true underlying problem by plain letters (f) and the corresponding approximating objects in our algorithm with hats ($\hat{f}$) if there is a risk of confusion; however, we omit the hat when there is no such risk, e.g. when denoting the vector $\mathbf{f}$ that approximates the function $f(\cdot)$.

A.1 General dynamic-programming problem

Consider a dynamic-programming problem with state vector $\mathbf{x} = (x_1, \ldots, x_n) \in X \subset \mathbb{R}^n$. We place the $c \leq n$ continuous state variables (e.g. assets) in the first positions of the state vector and assume that $x_i \in X_i$ for $i = 1, \ldots, c$, where $X_i \subset \mathbb{R}$ are closed intervals. There are $d = n - c$ discrete state variables with $x_j \in X_j = \{1, 2, \ldots, N_j\}$ for $j = c+1, \ldots, n$. The control vector is $\mathbf{u} \in U \subset \mathbb{R}^{N_u}$, whose dimension N_u may be larger, equal, or smaller than n. The feasible set for the control is given by a correspondence $\Gamma : X \rightrightarrows U$. The agent faces a return function $J : X \times U \to \mathbb{R}$ and discounts the future with the factor $\beta \in (0, 1)$. The law of motion is probabilisitic: The state at $t+1$, which we denote by $\mathbf{x}'$, is distributed according to a parametric distribution $f(\mathbf{x}'|\mathbf{z})$, thus $f : X \times Z \to \mathbb{R}_0^+$. Here, $\mathbf{z} \in Z \subset \mathbb{R}^{N_z}$ is a vector of moments for the c continuous state variables

The moments, in turn, depend on the state and the chosen control through a function $\mathbf{z} = z(\mathbf{x}, \mathbf{u})$, where $z : X \times U \to Z$. The Bellman Equation characterizing the value

function $W : X \to \mathbb{R}$ of the underlying problem is

$$W_{t-1}(\mathbf{x}) = \max_{\mathbf{u}\in\Gamma(\mathbf{x})} \left\{ J(\mathbf{x},\mathbf{u}) + \beta \int W_t(\mathbf{x}')\mathrm{d}f(\mathbf{x}'|z(\mathbf{x},\mathbf{u})) \right\}. \tag{35}$$

For finite-horizon problems, we have a function $W_T(\cdot)$ given at the final period T and iterate backward on (35) for $t = T-1,\ldots,1$. In infinite-horizon problems, the time subscript on the value functions in (35) is dropped and the Bellman Equation characterizes a fixed point $W(\cdot)$ of the finite-horizon recursions.

A.2 General algorithm

We first give a brief **overview of our algorithm** in the default setting, that is under infinite horizon and an approximation by first and second moments. We will then give further explanations and recommendations and discuss alternative choices (including the finite-horizon case) in the following subsection.

1. Place **grids** $\{\hat{x}_j^{(i)}\}_{j=1}^{N_i}$ with N_i elements on each of the continuous dimensions $i = 1,\ldots,c$. Denote by $\hat{X}$ the set of points on the resulting n-dimensional Cartesian grid, which has $N \equiv \prod_{i=1}^{n} N_i = |\hat{X}|$ elements (including now also the discrete dimensions). Furthermore, we denote by $\hat{x} \in \hat{X}$ a typical grid point and by $\mathbf{X} \in \mathbb{R}^{N\times n}$ the matrix that lists in each row the values that the state variables take on this grid.

2. **Law-of-motion/return-function approximation** (optional step). Create a function $\hat{z}(\mathbf{x},\mathbf{u})$ that approximates the law of motion $z(\cdot)$. This can be done i) by writing out flow variables as $x\Delta t$ and then dropping the lowest-order terms or ii) approximating controls by $u_i = \bar{u}_i + \Delta u_i$, taking a Taylor expansion of the law of motion (and possibly the return function) in the $\{\Delta u_i\}$ and keeping only the highest-order terms. In the value-function-iteration loop, then use the previous guess for the policy as $\bar{u}_i$.

3. **Modified Tauchen step**. Specify a probability distribution $\hat{f}(\hat{x}|\mathbf{z})$, $\hat{f} : \hat{X}\times Z \to \mathbb{R}_0^+$, over the grid $\hat{X}$ that approximates the continuous-support distribution $f(\mathbf{x}|\mathbf{z})$.

4. **Linear moment approximation**: For each fixed state $\hat{x} \in \hat{X}$ (at time t), we now approximate the distribution $\hat{f}$ over $\mathbf{x}'$ (at time $t+1$) given a fixed control vector $\mathbf{u}$ as follows:

   (a) Choose an **expansion point** $\bar{z}(\hat{x}) \in \mathbb{R}^{N_z}$ for the moments $\mathbf{z}$ at each grid point $\hat{x}$. This pins down the *reference distribution* $\mathbf{f}^{(0)}(\hat{x}) \equiv \hat{f}(\hat{x}|\bar{z}(\hat{x})) \in \mathbb{R}^N$ at each grid point.

   (b) For each moment z_l in $\mathbf{z}$, $l \in \{1, \ldots, N_z\}$, approximate the **derivative/**gradient of $\hat{f}(\cdot)$ in z_l evaluated at the expansion point $\bar{z}(\hat{x})$ and denote it by the vector $\mathbf{d}^{(z_l)}(\hat{x}) \in \mathbb{R}^N$.

   Combine (a) and (b) to obtain the following Taylor approximation for $\hat{f}$ given a choice $\mathbf{u} \in \hat{\Gamma}(\hat{x})$:

$$\mathbf{f}(\hat{x}, \mathbf{u}) = \mathbf{f}^{(0)}(\hat{x}) + \sum_{l=1}^{N_z} \big[z_l(\hat{x}, \mathbf{u}) - \bar{z}_l(\hat{x})\big] \mathbf{d}^{(z_l)}(\hat{x}) \qquad \in \mathbb{R}^N. \tag{36}$$

   To obtain the transition matrix on the discretized state space, we now stack controls into a matrix $\mathbf{U} = [\mathbf{u}(\hat{x})^T]_{\hat{x} \in \hat{X}} \in \mathbb{R}^{N \times N_u}$, each row containing the control vector for one Cartesian grid point. Similarly, create $N \times N$ matrices $\mathbf{F}^{(0)} \equiv [\mathbf{f}^{(0)}(\hat{x})^T]_{\hat{x} \in \hat{X}}$ for the reference distributions and $\mathbf{D}^{(z_l)} = [\mathbf{d}^{(z_l)}(\hat{x})^T]_{\hat{x} \in \hat{X}}$ for the derivatives, for all $l \in \{1, \ldots, N_z\}$. The transition matrix of the approximating Markov chain, given a control matrix $\mathbf{U}$, can be written as

$$\mathbf{F}(\mathbf{U}) = \mathbf{F}^{(0)} + \sum_{l=1}^{N_z} \big[z_l(\mathbf{X}, \mathbf{U}) - \bar{z}_l(\mathbf{X})\big] \circ \mathbf{D}^{(z_l)} \qquad \in \mathbb{R}^{N \times N}, \tag{37}$$

   where $z_l(\mathbf{X}, \mathbf{U})$ and $\bar{z}_l(\mathbf{X})$ are understood to be the $N \times 1$ vectors that result from applying the functions $z_l(\cdot)$ and $\bar{z}_l(\cdot)$ on each row of the input matrices separately.

5. **Ensuring positive transition probabilities**. For each grid point $\hat{x}$, specify an *admissible set* $\hat{\Gamma}(\hat{x})$ such that all elements of $\mathbf{f}(\hat{x}, \mathbf{u})$ in (36) are positive given any $\mathbf{u} \in \hat{\Gamma}(\hat{x})$.

6. **Value-function iteration loop**: Specify an initial guess $\mathbf{w}^{(0)} \in \mathbb{R}^N$ for the value function $W(\cdot)$ evaluated at the Cartesian grid points in $\hat{X}$ and fix a convergence criterion $\epsilon > 0$ (small). Then, for $s = 0, 1, \ldots$, solve the following (approximate) Bellman Equation:

$$\mathbf{w}^{(s+1)} = \max_{\mathbf{U} \in \hat{\Gamma}} \left\{ J(\mathbf{X}, \mathbf{U}) + \beta \Big(\mathbf{F}^{(0)} \mathbf{w}^{(s)} + \sum_{i=l}^{N_z} [z_l(\mathbf{X}, \mathbf{U}) - \bar{z}_l(\mathbf{X})] \circ [\mathbf{D}^{(z_l)} \mathbf{w}^{(s)}] \Big) \right\}, \tag{38}$$

where $\max$ is understood in the sense that each row $\mathbf{u}(\hat{x})^T$ of $\mathbf{U}$ is chosen such that the criterion for grid point $\hat{x}$ (the respective element of the vector in the curly brackets) is maximized. Denote by $\mathbf{U}^{*(s+1)}$ the optimal control matrix at iteration s. Once the value function converged, i.e. once $||\mathbf{w}^{(s+1)} - \mathbf{w}^{(s)}|| \leq \epsilon$ at iteration $s = s^*$, store the converged solution $\mathbf{w}^* = \mathbf{w}^{(s+1)}$ and $\mathbf{U}^* = \mathbf{U}^{*(s+1)}$ as well as the transition matrix $\mathbf{F}^*$ associated with the optimal control, which is obtained by setting $\mathbf{U} = \mathbf{U}^*$ in (37).[22]

7. **Distribution iteration loop**: Fix some initial distribution $\mathbf{g}_0 \in \mathbb{R}^N$, where $g_{0,i} \geq 0$ for all i and where $\sum_{i=1}^N g_{0,i} = 1$. Then, iterate for $s = 1, 2, \ldots$ on

$$\mathbf{g}_{s+1} = (\mathbf{F}^*)^T \mathbf{g}_s$$

until convergence to the stationary distribution $\mathbf{g}^*$.

8. **Sampling paths** $\{\hat{x}_t\}_{t=1}^S$ from the approximating Markov chain. Fixing an initial $\hat{x}_0 \in \hat{X}$, for $t = 0, 1, \ldots, S-1$ draw $\hat{x}_{t+1}$ from the probability distribution that is given by the row of $\mathbf{F}^*$ that corresponds to state $\hat{x}_t$.

A.3 Details and recommendations

We now discuss details on each of the steps in the above algorithm and give some recommendations. We start by explaining how to adapt the algorithm to a finite horizon.

[22] We opt for the standard choice $||\mathbf{x}|| = \max_{i \in \{1,\ldots,N\}} \{x_i\}$, but any norm on $\mathbb{R}^N$ is legitimate here. Also other convergence criteria here, e.g. convergence in policy functions or Euler-equation errors.

Finite-horizon case. The algorithm above can be adapted to finite-horizon problems making only minor adjustments. In Step 6 (value-function iteration loop), the terminal value function $\mathbf{w}^{(T)}$ is exogenously given by the problem and the recursions in (38) now run for $s = T, T-1, \ldots, 0$. Optimal controls $\mathbf{U}^{*(t)}$ and associated transition matrices $\mathbf{F}^{*(t)}$ have to be stored in each iteration. In Step 7 (distribution iteration loop), the initial distribution $\mathbf{g}^{(0)}$ is given by the problem and the recursions run from $t = 1$ to T; Step 8 (drawing paths) has to be adjusted analogously.

Grids. Grids may be linearly spaced, but need not be. As always, it is advisable to use many grid points in areas where shocks are small and/or policies are expected to vary a lot. For shocks that are proportional to the level of a variable, we recommend logarithmic grids.

Distribution $f(\cdot)$ and moments $\mathbf{z}$.

1. **Shape**. While other choices are covered by our algorithm, our recommended choice for the distribution $f(\cdot)$ is a multivariate normal distribution over the continuous dimensions, $\mathcal{N}(\mathbf{m}, \mathbf{V})$. $\mathbf{m} \in \mathbb{R}^c$ is the first moment (mean) vector and and $\mathbf{V} \in \mathbb{R}^{c \times c}$ is the covariance matrix. The moment vector is $\mathbf{z} = [\mathbf{m}, triag(\mathbf{V}))]$, where the operator $triag$ picks off the upper triagonal of the matrix, thus omitting the repeated entries below the diagonal of the symmetric matrix $\mathbf{V}$. A first advantage of the normal distribution is that it is highly tractable; a second advantage is that it is the only distribution that ensures consistency of the distributional shapes when we make time increments smaller. This is the case for the same reason why Brownian Motion in continuous time has normal increments: If we split periods in always shorter subperiods and assume independent shocks, then by the central limit theorem the sum over the shocks will tend to a normal distribution.

2. **Choice of moments**. Related to the last point, we recommend by default to use *all* means and (co)variances in the continuous dimensions as elements of the moment vector $\mathbf{z}$. However, our algorithm also covers the case in which $\mathbf{z}$ only covers a subset of moments, e.g. when the agent can only affect the mean (but not the variance of f); we did so in the basic consumption-savings problem where we chose savings $s = \mathbf{z}$ as one parameter that affected both the conditional mean and variance of future assets, a'. What is key (and should always be checked) when restricting $\mathbf{z}$ to a subset of moments is that the resulting approximating distributions $\mathbf{f}(\cdot)$ indeed give a

good approximation of f, at least in the first and second moment. We note here that choosing the moments as the mean and (co)variance matrix has the advantage that the approximating distributions in (36) conserve the variance of the underlying distribution, at least to a first-order approximation, as the following proposition shows:

Proposition A.1 (Mixtures first-order accurate for mean and variance) *Let X_i, $i \in \{1, \ldots, n\}$, be a sequence of random variables with mean $\mu_i = \mathbb{E}[X_i]$ and variance $\sigma_i^2 = \mathbb{E}[X_i - \mu_i]^2$. Let X be a random variable that is drawn from a mixture with weights $\{w_i\}_{i=1}^n$ of these n distributions, where $w_i \geq 0$ and $\sum_{i=1}^n w_i = 1$. Then:*

$$\mu = \mathbb{E}[X] = \sum_{i=1}^{n} w_i \mu_i, \tag{39}$$

$$\sigma^2 = \mathbb{E}[X - \mu]^2 = \sum_{i=1}^{n} w_i \sigma_i^2 \qquad \text{if } \mu_i = \mu \quad \forall i \in \{1, \ldots, n\}, \tag{40}$$

$$\sigma^2 = \mathbb{E}[X - \mu]^2 = \sum_{i=1}^{n} w_i (\Delta_i)^2 + \sum_{i=1}^{n} w_i \sigma_i^2 \qquad \text{where } \Delta_i = \mu_i - \mu. \tag{41}$$

(Proof to be completed, follows from algebra). The proposition says that approximating mixtures get the mean right always and the variance in case that the means of all distributions coincide. If both mean and variance vary across the mixed distributions, then the approximation error for the variance goes to zero as we make the grid finer and thus $\Delta_i \to 0$.

Importantly, we note here that first-order accuracy is *not* guaranteed for other choices of the moments than mean and variance. For example, if we parameterized the univariate standard normal distribution using the standard deviation σ instead of the variance σ^2, the linear approximations for $\mathbf{f}$ would have first-order approximation errors in terms of the second moment.[23]

[23] One may be tempted to just choose the controls themselves as the moments, i.e. to set $\mathbf{z} = \mathbf{u}$. However, this often leads to problems, as the following example shows. In the portfolio problem, this would mean to parameterize the distribution by consumption c and the portfolio share x in the portfolio problem. However, this makes the quadratic terms in x disappear in the first-order condition for x, thus leading to a bang-bang solution for x^*. This is an example in which an inappropriate choice of moments leads to a bad approximation of the distribution and to counterfactual properties for the solution.

3. The law of motion for the **discrete state** variables is, by default, given by exogenous Markov transition matrices. However, our algorithm can easily be generalized to the case in which the controls affect also the transition matrices in the discrete dimensions.

Expansion point $\bar{z}$. For mean moments, we recommend to set the expansion points on the grid points themselves, i.e. to choose $\bar{z}_l(\hat{x}) = \bar{m}_i(\hat{x}) = \hat{x}_i$ for a mean moment l and the corresponding dimension i. This choice implies that the *drift* (i.e. the expected time increment) is zero under the reference distribution (in line with what analogous continuous-time algorithms do). This has the advantage that the expansion point is always a good one as we let the time increment go to zero (at least for continuous laws of motion). However, we note that the algorithm is still well-defined for other choices of $\bar{m}_i$. Unlike drifts, reference variances $\bar{v}_{ii}(\hat{x})$ should be positive and the matrix $\bar{v}(\hat{x})$ should be positive definite in order to yield proper probability distributions for the approximating Markov chains.

Admissible set $\hat{\Gamma}$. The admissible set should be chosen such that the approximating vectors $\mathbf{f}(\cdot)$ in (36) are proper probability distributions, i.e. that all elements of $\mathbf{f}$ are non-negative and sum up to 1. If a control affects only the mean of one variable, this can usually be achieved by restricting the control such that the state cannot move farther than one grid point up or down (in the continuous dimensions) in one time period and by using upwind gradient approximations (see below). For example, in the basic consumption-savings problem we restricted the choice of savings to the interval between adjacent grid points, i.e. we set $\hat{\Gamma}(\hat{x}_j) = [\hat{x}_{j-1}, \hat{x}_{j+1}]$ at grid point $\hat{x}_j$. We then construct forward and backward gradients such that $\hat{f}$ is a convex combination of the distributions that are obtained when choosing $s \in \{\hat{x}_j, \hat{x}_{j+1}\}$ (for a saver) or $s \in \{\hat{x}_j, \hat{x}_{j-1}\}$ (for a dissaver).[24]

Approximating f **by** $\hat{f}$. In principle, any reliable method that discretizes a continuous-support random variable can be used here. Some choices are as follows:

1. If there is only one continuous dimensions the ($c = 1$), one can approximate $\hat{f}$ by simply calculating the probabilities under the true f of landing in a bin corresponding

[24]Positive probabilities are necessary so that i) the convergence proofs for Markov-chain control problems hold and ii) the calculation of the distribution (Step 7) and simulations (Step 8) in our algorithm work. However, we note here the Taylor approximation of the transition probability vector in (36) is valid even if probabilities are negative, implying that the value-function-iteration loop can still succeed and return good approximations even when negative probabilities occur. In our examples, however, we found that the algorithm often fails to converge once negative probabilities become too large in absolute value.

that corresponds to a grid point as[25]

$$\hat{f}(\hat{x}|\mathbf{z}) = F\big((\hat{x}_{j+1} + \hat{x}_j)/2|\mathbf{z}\big) - F\big((\hat{x}_j + \hat{x}_{j-1})/2|\mathbf{z}\big), \tag{42}$$

where $F(\cdot|\mathbf{z})$ denotes the cdf of $f(\cdot\,|\mathbf{z})$ for a fixed $\mathbf{z}$. At the lower (upper) grid margins, the upper and lower bin bounds have to be replaced by $\pm\infty$. This procedure is used in our function `TransMat.m`. The method can also be used in the multivariate case, i.e. $c > 1$, if shocks are independent across the continuous dimensions; one can then obtain the marginal probability for each dimension separately as in (42) and then multiply to obtain the joint probability.

2. Integrate the density (or approximate the integral by quadrature) in the multi-variate case, analogous to the integration in (42) for the one-dimensional case.

3. Monte-Carlo simulation: Draw the underlying shocks using a random-number generator and count how many draws of $\mathbf{x}'$ fall into an area representing a grid point. This method has the advantage that it is simple, but has the drawback that it introduces additional sampling error.

Again at this point, it is advantageous to work with normal distributions. In the portfolio choice problem, for example, assuming log-normal distributions for the safe and risky return leads to a asset distribution that has no closed form for the density, being a sum of two log-normals, thus complicating the calculation of probabilities. Under our bivariate normal assumption, however, the sum of the two returns is again normally distributed and probabilies are easily computed.[26]

Derivative vectors $\mathbf{d}^{(z_l)}$. We recommend to approximate the derivatives using *difference quotients*, as they are common in numerical work. For each moment z_l, $l \in \{1, \ldots, N_z\}$, specify increment vectors $\Delta_l^+, \Delta_l^- \in \mathbb{R}^{N_z}$ that are zero except a positive entry on the lth position. Then the difference quotients are defined as:

[25] This is the method used by Tauchen (1986).

[26] We note here that the normal distribution has similar shape to the log-normal once time periods become small, thus the approximation error is small even if the true return distribution was log-normal.

1. *forward*:
$$\mathbf{d}^{(z_l)+}(\hat{x}) = \frac{\hat{f}\big(\hat{x}|\bar{z}(\hat{x}) + \Delta_l^+\big) - \hat{f}\big(\hat{x}|\bar{z}(\hat{x})\big)}{||\Delta_l^+||}$$

2. *backward*:
$$\mathbf{d}^{(z_l)-}(\hat{x}) = \frac{\hat{f}\big(\hat{x}|\bar{z}(\hat{x})\big) - \hat{f}\big(\hat{x}|\bar{z}(\hat{x}) - \Delta_l^-\big)}{||\Delta_l^-||}$$

3. *upwind*: use the forward quotient if $z_l(\hat{x}, \mathbf{u}) > \bar{z}_l(\hat{x})$ and the backward quotient otherwise.

4. *centered*:
$$\mathbf{d}^{(z_l)c}(\hat{x}) = \frac{\hat{f}\big(\hat{x}|\bar{z}(\hat{x}) + \Delta_l^+\big) - \hat{f}\big(\hat{x}|\bar{z}(\hat{x}) - \Delta_l^-\big)}{||\Delta_l^+|| + ||\Delta_l^-||}$$

For mean moments, i.e. if $z_l = m_i$ for dimension i, then we recommend to choose the increments Δ_i^+ and Δ_i^- as the distance to the adjacent grid points. When one restricts the feasible set $\hat{\Gamma}$ such that the state cannot move more than one grid point away, then the upwind approach boils down to taking convex combinations of the distributions at the three adjacent grid points. This is the approach we followed in the basic consumption-savings problem. It has the advantage that it automatically guarantees that the approximating distributions are proper probability distributions. The other three approaches (forward, backward, centered) have the advantage that they only require to calculate *one* instead of *two* gradients, and only one FOC (not two) has to be evaluated for the control(s). An advantage of the centered quotient is that it provides a second-order accurate estimate of the derivative at $\hat{x}$, which is not the case for the other approaches. Thus, we recommend the centered quotient over the backward or forward quotients.

In principle, one could also approximate the derivatives based on the analytical derivatives of the pdf $f(\cdot)$ in the moments. A drawback of this approach is, however, that it is *not* guaranteed that the approximating probability distributions $\mathbf{f}$ sum to one, since the components of $\hat{\mathbf{d}}_i$ need not sum to zero. Note that this normalization property is ensured, however, for the difference quotients 1.-4. since we use proper probability distributions when constructing them.

Finding the optimal control. In many of our examples, we could find a closed-form solution for the optimal control $\mathbf{u}_{s+1}^*$ in the approximating Bellman Equation (38) using

first-order conditions. We note that this often occurs although there is no closed-form solution in the true Bellman Equation (35). Closed-form expressions speed up the algorithm considerably since they avoid costly nonlinear-solver or root-finding routines. The reason for these simplifications is that the approximation drops higher-order effects that controls have on the distributions, as is the case in the continuous-time HJB.

Law-of-motion approximation. It can be the case that (even after using the linear-moment approximation) no closed-form solution for the optimal control obtains in the approximate Bellman Equation, while the corresponding continuous-time HJB does yield a closed-form solution. We found this to be the case in the portfolio problem; the reason being that second-order terms are present in the moment function that drop out in the continuous-time limit.

(Modified) policy function iteration. The usual principles for dynamic-programming algorithms in general and for Markov-chain-approximation algorithms in particular apply to the approach we propose. We recommend Phelan & Eslami (2021), an excellent introduction to Markov-chain approximation methods in economics; they study the relative efficiency of policy-function-iteration algorithms, however in the context of continuous-time problems. *Policy function iteration* means to find (at each step of value-function iteration) the value that obtains from applying the current policy guess forever (this is also sometimes called the "Howard improvement algorithm"). *Modified policy function iteration*, in turn, means that the current policy is only applied for $K \geq 1$ periods instead of forever. Formally, consider the following equation that updates a value function applying the policy found in step s. Initiating with the value-function guess from the last iteration, $\mathbf{v}^{(s,0)} = \mathbf{w}^{(s-1)}$, compute for $k = 1, \ldots, K$:

$$\mathbf{v}^{(s,k+1)} = J(\mathbf{X}, \mathbf{U}^{*(s)}) + \beta \mathbf{F}(\mathbf{U}^{*(s)}) \mathbf{v}^{(s,k)}, \tag{43}$$

i.e. apply the policy $U^{*(s)}$ for k periods. Finally, use the obtained value as the new guess, i.e. set $\mathbf{w}^{(s+1)} = \mathbf{v}^{(s,K+1)}$. Note here that value function iteration obtains as a special case when setting $K = 1$ and (classical) policy function iteration obtains when letting $K \to \infty$; the value function for the latter case can also be computed as the solution to the linear system

$$\big[\mathbf{I} - \beta \mathbf{F}(\mathbf{U}^{*(s)})\big] \mathbf{w}^{(s)} = J(\mathbf{X}, \mathbf{U}^{*(s)}), \tag{44}$$

where $\mathbf{I}$ is the $N \times N$ identity matrix. This equation can be solved using a sparse linear solver (as is done, e.g., by Achdou et al. (2022) for an analogous continuous-time problem). As Phelan & Eslami (2021), we find that *modified policy function iteration* yielded the largest speed gains in our applications and that policy-function iteration was only advantageous in low-dimensional applications ($n \leq 2$). The reason is that once the number of transitions between states increases, approximating the solution to (44) by K successive iterations as in (43) becomes more efficient than solving a large system of equations.

Comparison to continuous-time HJB. We can re-write Eq. **??** for the converged solution $\mathbf{w}^*$, spelling out the mean and (co)variance components of the moment vector $\mathbf{z}$:

$$\begin{aligned}\big(\mathbf{I} - \beta\mathbf{F}^{(0)}\big)\mathbf{w}^* = \max_{\mathbf{U}\in\hat{\Gamma}} \Bigg\{ & J(\mathbf{X},\mathbf{U}) + \sum_{i=1}^{c} [m_i(\mathbf{X},\mathbf{U}) - \bar{\mathbf{m}}_i] \circ [\mathbf{D}^{(m_i)}\mathbf{w}^*] \\ & + \sum_{i=1}^{c}\sum_{j=i}^{c} [v_{ij}(\mathbf{X},\mathbf{U}) - \bar{\mathbf{v}}_{ij}] \circ [\mathbf{D}^{(v_{ij})}\mathbf{w}^*] \Big) \Bigg\},\end{aligned} \tag{45}$$

which compares to the continuous-time HJB

$$\rho V(\mathbf{x}) = \max_{\mathbf{u}\in\Gamma(\mathbf{x})} \left\{ J(\mathbf{x},\mathbf{u}) + \sum_{i=1}^{c} m(\mathbf{x},\mathbf{u}) V_i + \tfrac{1}{2}\sum_{i=1}^{c}\sum_{i=1}^{c} v_{ij}(\mathbf{x},\mathbf{u}) V_{ij} \right\} + \text{jump terms} \tag{46}$$

where $\rho = \ln\beta \simeq 1-\beta$ is the continuous-time discount rate corresponding to the discount factor β, V_i is partial derivative in dimension $i = 1,\ldots,c$ and V_{ij} denote the second partial derivatives in the continuous dimensions. We see that the discrete-time Bellman equation (45) corresponds closely to (46). In the pseudo-Hamiltonian in the max-operator, the terms $D_m V$ take the role of the first value-function derivative, V_i, and $D_v V$ takes the role of the Hessian, V_{ij}. The term $(\mathbf{I} - \beta\mathbf{F}^{(0)})\mathbf{w}^*$ represents discounting (compare to ρV) and *jump terms* stemming from transitions in the discrete dimensions (contained in $\mathbf{F}^{(0)}$).

B Tensor approach under independent shocks

When shocks are independent across dimensions, exploiting the tensor structure of transition probabilities can to lead major speed gains. The higher the dimensionality of the

state space, the larger these gains will be. For example, in our altruism application with $n = 4$ dimensions and grid size $(N_1, N_2, N_3, N_4) = (41, 41, 2, 2)$, the value-function iteration loop can be sped up by a factor 70 (from 17.03 to 0.24 seconds) and the distribution iteration loop by a factor 7 (from 0.28 to 0.04 seconds).[27] What is the reason for these speed gains? In a nutshell: It is fast to apply simple transition matrices sequentially dimension by dimension, while it takes long to create and apply large transition matrices on the full Cartesian state space. In this appendix, we draw on tensor notation from the literature that is particularly well-suited to compactly describe the necessary numerical operations. We also compare this notation to the commands in our Matlab code to illustrate these concepts, which may not be familiar to many economists.

Setup. Consider again a general problem with n dimensions and $c \leq n$ continuous state variables. Assume that shocks in the different dimensions are independent from each other. This directly implies that all covariances are zero, i.e. that $\sigma_{ij} = 0$ for all $i \neq j$. Assume that the transition density in the continuous dimensions under the true model is given by functions $f_i(x_i'|x_i; \mu_i, \sigma_{ii})$, for $i = 1, \ldots, c$, and that the transition probabilities in the discrete dimensions are given by $\pi_j(x_j', x_j)$ for $j = c+1, \ldots, n$.[28] By independence, the conditional probability density function (pdf) on the n-dimensional state space is given by the *tensor product*

$$f(\mathbf{x}'|\mathbf{x}) = \prod_{i=1}^{c} f_i(x_i'|x_i; \mu_i, \sigma_{ii}) \prod_{j=c+1}^{n} \pi_j(x_j'|x_j). \tag{47}$$

Notation. To keep track of values, distributions and the like, we will use n-dimensional arrays (or *tensors*) instead of the $N \times 1$ vectors that we used before (where again $N = \prod_{i=1}^{n} N_i$). Following the notation in De Lathauwer et al. (2000), we will denote tensors with calligraphic letters (e.g. $\mathcal{A}$). For example, the approximate value function is now an n-dimensional array (or *order-n* tensor) $\mathcal{W} \in \mathbb{R}^{N_1 \times N_2 \times \ldots N_n}$ instead of the previous vector $\mathbf{w} \in \mathbb{R}^N$.[29] We index the elements of the tensor$\mathcal{A}$ by $A_{j_1, j_2, \ldots, j_n}$, analogously to ma-

[27] There is an additional gain of 1.5 seconds with the tensor approach before the loop when the transition matrices have to be created.

[28] Again, our approach can be generalized to the case in which the controls affect also the transitions in the discrete dimensions; we again omit this case here for brevity.

[29] The relationship between the two ways of denoting values is $\mathbf{w} = vec(\mathcal{W})$; we will introduce the vec-

trix indexing A_{j_1,j_2}, and write $\mathcal{A} = [A_{j_1,j_2,\ldots,j_n}]$ to collect single entries to a tensor over a range of indeces. In Matlab, a 4-dimensional (4D) array, for example, is allocated by the command `A = zeros(N1,N2,N3,N4)`, and an element is retrieved by indexing `A(j1,j2,j3,j4)`. We denote by $\{\mathbf{F}^{(0,i)}\}_{i=c+1}^{n}$, the $N_i \times N_i$ transition matrices in the discrete states, which are exogenously given; entry $F^{(0,i)}_{j_i,j'_i}$ gives the probability of transitioning from state j_i to state j'_i. Also, denote by $\{\mathbf{F}^{(0,i)}\}_{i=1}^{c}$ the $N_i \times N_i$ transition matrices on the grid for continuous dimension i that result from the reference drifts and variances, $\{\bar{m}_i, \bar{v}_{ii}\}_{i=1}^{c}$.

Mode products. We first define an operation "tensor times matrix" that gives us a notion of taking expectations over transitions in one dimension i, keeping the states in all other dimensions fixed:

Definition 1 (i-mode product) *Given a tensor $\mathcal{A} \in \mathbb{R}^{N_1 \times \cdots \times N_i \times \cdots \times N_n}$ and a matrix $\mathbf{F}^{(i)} \in \mathbb{R}^{M_i \times N_i}$, define the* i-mode product $\mathcal{A} \times_i \mathbf{F}^{(i)} \in \mathbb{R}^{N_1 \times \cdots \times M_i \times \cdots \times N_n}$ *by*

$$\left(\mathcal{A} \times_i \mathbf{F}^{(i)}\right)_{j_1,\ldots,j_i,\ldots,j_n} \equiv \sum_{j'_i=1}^{N_i} F^{(i)}_{j_i j'_i} A_{j_1,\ldots,j'_i,\ldots,j_n}. \tag{48}$$

Note that the sum in (48) weighs values by the transition probabilities from state j_i to all other states j'_i of dimension i. Thus, the mode-i product captures the operator $\mathbb{E}_{x'_i}[\cdot|x_i]$, as claimed before. For example, we can take an expectation of a value $\mathcal{W}'$ at $t+1$ before the shock in dimension i hits as $\mathcal{W}' \times_i \mathbf{F}^{(0,i)}$; this is the interpretation when moving *backward* in time. Similar to what is the case for for conventional Markov chains, it turns out that a mode product using the *transposed* transition matrix captures a notion of going *forward* in time (the so-called *adjoint property*). The i-mode product with the transposed transition matrix, $\mathcal{P} \times_i \mathbf{F}^{(i)^T}$, signifies that a distribution $\mathcal{P}$ over the n-dimensional state space is mapped forward in time, applying only the transitions in dimension i:

$$\left(\mathcal{P} \times_i (\mathbf{F}^{(i)})^T\right)_{j_1,\ldots,j'_i,\ldots,j_n} = \sum_{j_i=1}^{N_i} F^{(i)}_{j_i j'_i} P_{j_1,\ldots,j_i,\ldots,j_n}. \tag{49}$$

Here, we have stuck to the convention that j_i denotes positions at t and j'_i the position at

torization of tensors below.

$t+1$. In Matlab, we provide the function `MatrixTimesArray` that operationalizes the i-mode product: `MatrixTimesArray(F0i,W,i)` returns $\mathcal{W} \times_i \mathbf{F}^{(0,i)}$. The function uses Matlab's built-in matrix multiplication, which is very efficient.[30]

Obviously, the mode-i product is similar to a matrix product; it shares some, but not all of its **properties**:[31]

1. *Distributive*: $\mathcal{A} \times_i (\mathbf{F} + \mathbf{G}) = \mathcal{A} \times_i \mathbf{F} + \mathcal{A} \times_i \mathbf{G}$ (obvious)

2. "*Matrix chain rule*": $\mathcal{A} \times_i (\mathbf{FG}) = (\mathcal{A} \times_i \mathbf{G}) \times_i \mathbf{F}$ (Property 3 in DeLathauwer et al.)

3. *Commutative* across dimensions: For $i \neq j$, $(\mathcal{A} \times_i \mathbf{F}) \times_j \mathbf{H} = (\mathcal{A} \times_j \mathbf{H}) \times_i \mathbf{F} = \mathcal{A} \times_i \mathbf{F} \times_j \mathbf{H}$. (Property 2 in De Lathauwer et al., SIAM 2000).

The first two rules are intuitive and hold for precisely the same reason as they do in conventional matrix algebra.[32] The third rule is particular to the tensor case, however: In our context, commutativity across dimensions means that it does not matter in which order we carry out the state transitions in the different dimensions (note here that this is true only since we assumed that transitions are independent across dimensions).

Following Bader & Kolda (2006), we also introduce a notation for sequences of mode products; this will give us a tensor notation for calculating expectations with respect to transitions in *all* dimensions, i.e. the operator $\mathbb{E}_{\mathbf{x}'}[\cdot|\mathbf{x}]$, as we will show below:

Definition 2 (Sequence mode product) *Let $\mathcal{A} \in \mathbf{R}^{N_1 \times \cdots \times N_n}$ be an order-n tensor and let $\{\mathbf{F}\} = \{\mathbf{F}^{(i)}\}_{i=1}^n$ be a sequence of matrices satisfying $\mathbf{F}^{(i)} \in \mathbb{R}^{M_i \times N_i}$. Then, we define the* sequence mode product *as*

$$\mathcal{A} \times \{\mathbf{F}\} \equiv \mathcal{A} \times_1 \mathbf{F}^{(1)} \times_2 \mathbf{F}^{(2)} \cdots \times_n \mathbf{F}^{(n)} \qquad \in \mathbb{R}^{M_1 \times \cdots \times M_n}. \tag{50}$$

[30] The function reshapes and permutes the array `W` if necessary. The `reshape` commands in Matlab are computationally cheap, whereas the `permute` commands are computationally more demanding since they require the array to be re-written in memory. Permutes are thus only carried out if necessary, that is, only when both $n > 1$ and $i < n$. For the case $i = 1$, matrix premultiplication (after a reshape) is used, wherease matrix post-multiplication (after reshape) is used in the case $i = n$. In our applications, the cost of permuting the array is easily outweighed by the gains from the tensor approach.

[31] All of these properties are stated under the assumption that the matrices and tensors are conformable.

[32] One way of deriving these rules is actually to convert the order-n tensor $\mathcal{A} \in \mathbb{R}^{N_1 \times \cdots \times N_n}$ to a $N_i \times (N_1 \ldots N_{i-1} N_{i+1} \ldots N_n)$ matrix —this operation is called "matricizing", "flattening" or "unfolding" the tensor, see Bader & Kolda— and then applying matrix algebra rules.

Similarly, we define the sequence mode product in all dimensions except i as

$$\mathcal{A} \times_{-i} \{\mathbf{F}\} \equiv \mathcal{A} \times_1 \mathbf{F}^{(1)} \cdots \times_{i-1} \mathbf{F}^{(i-1)} \times_{i+1} \mathbf{F}^{(i+1)} \cdots \times_n \mathbf{F}^{(n)}. \tag{51}$$

Similarly, a sequence mode product with the transposes of all transition matrices, which we will denote by $\mathcal{P} \times \{\mathbf{F}^{(0)^T}\}$, signifies that a distribution $\mathcal{P}$ is mapped forward in time, applying the transitions in all dimensions. In the main result of this appendix, we will shortly show that computing a sequence mode product is superior to a matrix product using a large $N \times N$ transition matrices on the Cartesian state space (both for updating expectations backwards and mapping distributions forward).

Vectorization. To do so formally, however, we first have to define the vectorization of a tensor. We do this in the obvious way by extending the vectorization of matrices. For order-2 tensors (matrices), we maintain the usual definition of vectorization, which is to concatenate all columns of the matrix in a long vector. For an order-3 tensor (which can be pictured as a sequence of matrices stacked behind each other on *pages*), we extend vectorization as is natural: We loop over all indices $j_3 = 1, \ldots, N_3$ of the third dimension (the pages), vectorize the matrices that obtain when fixing index j_3 in the tensor, and concatenate the N_3 resulting vectors to a large vector of length $N = N_1 N_2 N_3$. We can then generalize this operation recursively to higher dimension:

Definition 3 (Tensor vectorization) *Recursively, for an order-n tensor $\mathcal{A} \in \mathbb{R}^{N_1 \times \cdots \times N_n}$ define*

$$vec(\mathcal{A}) = [vec(\mathcal{A}_{\ldots 1}), vec(\mathcal{A}_{\ldots 2}), \ldots, vec(\mathcal{A}_{\ldots N_n})] \qquad \textit{for } n = 2, 3, \ldots, \tag{52}$$

where $\mathcal{A}_{\ldots j_n}$ denotes the order $(n-1)$ tensor that results when fixing index j_n in dimension n in $\mathcal{A}$, but letting all indeces vary over their entire range. The recursions start with the identity $vec(\mathcal{A}) = \mathcal{A} \in \mathbb{R}^{N_1}$ for an order-1 tensor (vector).

In (52), the order-$(n-1)$ tensors $\mathcal{A}_{\ldots i} \in \mathbb{R}^{N_1 \times \cdots \times N_{n-1}}$ are called the *slices* of the tensor $\mathcal{A}$ along dimension n. For $n = 3$, for example, the slices along dimension 3 are $N_1 \times N_2$ matrices. In Matlab, vectorization of a tensor (n-dimensional array) `A` is simply achieved by typing `A(:)`.[33] The ith slice of the tensor along the last dimension, $\mathcal{A}_{\ldots i}$, can be obtained

[33]The output from this command also reflects how Matlab actually stores the array, which is as a sequence

in by typing `A(:,:,:,i)` —here for a 4D array `A`—, which returns a 3D array.

We now have everything in place to convey the main message of this section: If shocks are independent, use sequence mode products on multi-dimensional arrays, not matrix multiplication with a large Kronecker matrix. This result is well-known in the literature (see Fackler (2019) and the references therein), but we state a proposition here to have all the relevant results in one place.

Proposition B.1 (Sequence mode product beats equivalent Kronecker product) *Let $\mathcal{W} \in \mathbf{R}^{N_1 \times \cdots \times N_n}$ be an order-n tensor, $\{\mathbf{F}\} = \{\mathbf{F}^{(i)}\}_{i=1}^{n}$ a sequence of matrices satisfying $\mathbf{F}^{(i)} \in \mathbb{R}^{N_i \times N_i}$ for all i, and let $N \equiv \prod_{i=1}^{n} N_i$. Then the sequence mode product is equivalent to multiplication by the Kronecker product of the matrices $\{\mathbf{F}^{(i)}\}$, i.e.*

$$vec(\mathcal{W} \times \{\mathbf{F}\}) = \big(\underbrace{\mathbf{F}^{(1)} \otimes \mathbf{F}^{(2)} \cdots \otimes \mathbf{F}^{(n)}}_{\equiv \mathbf{F}^{kron} \in \mathbb{R}^{N \times N}}\big) vec(\mathcal{W}) = vec\Big(\big[\sum_{all\ \mathbf{j}'} W_{\mathbf{j}'} \prod_{i=1}^{n} F^{(i)}_{j_i, j_i'}\big]_{\mathbf{j}}\Big). \quad (53)$$

Furthermore,

1. ***(speed comparison)** The sequence product $\mathcal{W} \times \{\mathbf{F}\}$ requires $N(\sum_{i=1}^{n} N_i)$ multiplications (and additions), being a factor $N/(\sum_{i=1}^{n} N_i)$ faster than the matrix product $\mathbf{F}^{kron} vec(\mathcal{W})$, which requires N^2 multiplications (and additions).*

2. ***(order of sequence irrelevant)** The sequence product can be computed in any order over $i \in \{1, \ldots, n\}$, each ordering resulting in the same number of multiplications (and additions).*

of numbers in memory arranged in the order of `A(:)` (note here that Matlab uses the column-major format).

Proof: Observe that

$$\begin{aligned}
\mathcal{W} \times \{\mathbf{F}\} &= \mathcal{W} \times_1 \mathbf{F}^{(1)} \cdots \times_n \mathbf{F}^{(n)} \\
&= \left[\sum_{j_1'=1}^{N_1} F^{(1)}_{j_1 j_1'} \mathcal{W}_{j_1', j_2 \ldots, j_n} \right] \times_2 \mathbf{F}^{(2)} \cdots \times_n \mathbf{F}^{(n)} \\
&= \left[\sum_{j_2'=1}^{N_2} F^{(2)}_{j_2 j_2'} \sum_{j_1'=1}^{N_1} F^{(1)}_{j_1 j_1'} \mathcal{W}_{j_1' j_2' j_3 \ldots j_n} \right] \times_3 \mathbf{F}^{(3)} \cdots \times_n \mathbf{F}^{(n)} \\
&= \ldots \\
&= \left[\sum_{j_n'} \sum_{j_{n-1}'} \cdots \sum_{j_1'} F^{(n)}_{j_n j_n'} \ldots F^{(1)}_{j_1 j_1'} \mathcal{W}_{j_1', \ldots, j_n'} \right]
\end{aligned}$$

Vectorizing the tensor in the last line, one obtains the vector $\mathbf{F}^{kron} vec(\mathcal{W})$, thus justifying Eq. (53). To show Point 1. of the proposition, note that in the first equality above, each mode-i product requires NN_i multiplications (and additions), from which it follows that the sequence product requires $N(\sum N_i)$ multiplications (and additions); the statements on the number of multiplications (and additions) in the matrix product $\mathbf{F}^{kron} vec(\mathcal{W})$, where $\mathbf{F}^{kron}$, follow from $\mathbf{F}^{kron}$ being an $N \times N$ matrix. To show Point 2. of the proposition, note that the above derivation yields the identical result no matter which ordering of the sequence product we choose in the first equality; also, each mode-i product requires NN_i multiplications (and additions) irrespective of the order of multiplication. ■

Two remarks are in order:

Remark B.1 (Relevance of sequence-product ordering) *If at least one the matrices $\mathbf{F}^{(i)}$ were non-square, i.e. $M_i \neq N_i$ for some i, then there is an optimal ordering to compute the sequence product, which is given in Fackler (2019).*

Remark B.2 (Speed savings for sparse matrices) *If the matrices $\{\mathbf{F}^{(i)}\}$ are sparse with an average number of $\tilde{N}_i < N_i$ non-zero entries per row, then the gain factor in Point 2. of the proposition has to be replaced by $\tilde{N}/(\sum \tilde{N}_i)$, where we define $\tilde{N} \equiv \prod_{i=1}^n \tilde{N}_i$.*

Remark B.2 is proven following exactly the same steps as the proof of the proposition.

How large are the speed gains of Prop. B.1 in practice? In the example of the altruism model, the savings factor is $40^2 2^2/(2 \cdot 40 + 2 \cdot 2) \simeq 40 \cdot 2 = 80$, which is roughly in line

with the factor 70 we obtain in our code. In a more general example with c continuous dimensions with N_c grid points each and d discrete dimensions with N_d grid points each, we obtain a savings factor $N_c^c N_d^d/(c \cdot N_c + d \cdot N_d) \simeq N_c^{c-1} N_d^d/c$, which we note to grow very rapidly in the number and grid size of the continuous dimensions. For example, with $c = 3$ continuous dimension à 100 grid points and 2 discrete dimensions à 10 grid points (the total number of points on the Cartesian being $N = 10^8$, which is about what fits into the RAM of a conventional computer) the accelaration factor is$100^2 10^2/3 = 10^6/3$, i.e. a third of a million(!).

Calculating continuation values efficiently. We now return to the general control problem with independent shocks and show how sequence products can be applied there. To fix ideas, first consider an example with a 3D state space in which the agent controls the drifts in the two continuous dimensions. That is, we set $n = 3$, $c = 2$ and $\mathbf{z} = (m_1, m_2)$. The first-order approximation of the conditional density in the moments $\mathbf{z} = (m_1, m_2)$ is then

$$\begin{aligned} f(\mathbf{x}'|\mathbf{x}, m_1, m_2) =& f^{(1)}(x_1'|x_1, m_1) f^{(2)}(x_2'|x_2, m_1) \pi_3(x_3'|x_3) \\ \simeq& \underbrace{f^{(1)}(x_1'|x_1, \bar{m}_1) f^{(2)}(x_2'|x_2, \bar{m}_2) \pi_3(x_3'|x_3)}_{\text{transitions under reference drifts } (\bar{m}_1, \bar{m}_2)} \\ &+ \underbrace{(m_1 - \bar{m}_1) f_{m_1}^{(1)}(x_1'|x_1, \bar{m}_1) f^{(2)}(x_2'|x_2, \bar{m}_1) \pi_3(x_3'|x_3)}_{\text{adjustment for drift in dimension 1}} \\ &+ \underbrace{(m_2 - \bar{m}_2) f_{m_2}^{(2)}(x_2'|x_2, \bar{m}_2) f^{(1)}(x_1'|x_1, \bar{m}_1) \pi_3(x_3'|x_3)}_{\text{adjustment for drift in dimension 2}}, \end{aligned} \tag{54}$$

where $f_{m_i}^{(i)}$ denotes the partial derivative of $f^{(i)}(\cdot)$ in m_i. The first equality in (54) follows from the independence assumption; the following approximation is then as in (36) in the main text, but exploiting independence (the tensor structure). Importantly note that in each summand in the Taylor expansion,we only have to take partial derivatives in *one* of the functions $f^{(i)}$ while keeping the others fixed.

Moving on to the discretization, we now approximate the functions in (54) by vectors. The transition probability from the (Cartesian) grid point $\hat{x} \in \hat{X} \subset \mathbb{R}^3$ indexed by $\mathbf{j} =$

$[j_1, j_2, j_3]$ to grid point $\hat{x}' \in \hat{X}$ indexed by $\mathbf{j}' = [j_i']$ is

$$F_{[\mathbf{j},\mathbf{j}']}|(m_1, m_2) = \underbrace{F^{(0,1)}_{j_1 j_1'} F^{(0,2)}_{j_2 j_2'} F^{(0,3)}_{j_3 j_3'}}_{\text{transition under reference drifts}} + \underbrace{(m_1 - \bar{m}_1) D^{(m_1)}_{j_1,j_1'} F^{(0,2)}_{j_2,j_2'} F^{(0,3)}_{j_3,j_3'}}_{\text{adustment for drift in dim. 1}}$$
$$+ \underbrace{(m_2 - \bar{m}_2) D^{(m_2)}_{j_2,j_2'} F^{(0,1)}_{j_1,j_1'} F^{(0,3)}_{j_3,j_3'}}_{\text{adustment for drift in dim. 2}}, \tag{55}$$

where we denote by$\mathbf{F}^{(0,3)} = [\pi(\hat{x}^{(3)}_{j_3}, \hat{x}^{(3)}_{j_3'})] \in \mathbb{R}^{N_3 \times N_3}$ the transition matrix in the discrete dimension and by $\mathbf{F}^{(0,1)}$ and $\mathbf{F}^{(0,2)}$ the transition matrices in the continuous dimensions given the reference drifts $\bar{m}_1$ and $\bar{m}_2$. Also, $\mathbf{D}^{(m_i)} \in \mathbb{R}^{N_i \times N_i}$ is the derivative matrix in dimension $i = 1, 2$ with respect to m_i. In particular, the j_ith row of the derivative matrix, $\mathbf{D}^{(m_i)}_{j_i \cdot}$, tells us how the reference distribution in dimension i, (the j_ith row of the matrix $\mathbf{F}^{(0,i)}$) changes when the drift m_i changes marginally at the expansion point. Conveniently, the derivative matrices $\mathbf{D}^{(m_i)}$ can be constructed exactly as described in the main text for the one-dimensional case. Note that in (55), the matrix entries $D^{(i)}_{j_i,j_i'}$ play the role of the partial derivatives $f^{(i)}_{m_i}$ in (54) and the entries $F^{(0,i)\cdot}_{j_i,j_i'}$ play the role of the transition density $f^{(i)}(\cdot)$ under the reference drift.

Using (53) from Prop. B.1, we can now collect all grid points in (55) and write the continuation value $\mathcal{C} = \mathbb{E}[\mathcal{W}'|\cdot]$ at t given a value function $\mathcal{W}'$ at $t+1$ in tensor form:

$$\mathcal{C}(\mathcal{M}_1, \mathcal{M}_2) = \underbrace{\mathcal{W}' \times \{\mathbf{F}^{(0)}\}}_{\text{exp. value under reference drifts}} + (\mathcal{M}_1 - \bar{\mathcal{M}}_1) \circ \big(\underbrace{\mathcal{W}' \times_1 \mathbf{D}^{(m_1)} \times_2 \mathbf{F}^{(0,2)} \times_3 \mathbf{F}^{(0,3)}}_{\text{derivative of contin. value in } m_1} \big)$$
$$+ (\mathcal{M}_2 - \bar{\mathcal{M}}_2) \circ \big(\underbrace{\mathcal{W}' \times_2 \mathbf{D}^{(m_2)} \times_1 \mathbf{F}^{(0,1)} \times_3 \mathbf{F}^{(0,3)}}_{\text{derivative of continuation value in } m_2} \big), \tag{56}$$

where $\mathcal{M}_i, \bar{\mathcal{M}}_i \in \mathbb{R}^{N_1 \times \cdots \times N_n}$, $i = 1, 2$, are order-n tensors that collect the mean moments m_i and their expansion points $\bar{m}_i$ across the Cartesian grid and where $\circ$ denotes element-by-element multiplication of two tensors. (56) is called a *backward equation* since it tells us how to update backward in time.

Following the idea for the 3D example, we now generalize to the general n-dimensional case with a general moment vector $\mathbf{z} \in \mathbf{R}^{N_z}$. To do this, we first introduce some more notation. Denote by $i^*(l)$ the function that tells us the dimension that the moment z_l in $\mathbf{z}$

addresses, where $i^* : \{1, \dots, N_z\} \to \{1, \dots, n\}$. For example, if $\mathbf{z} = (m_1, m_2, v_2)$, then we have $i^*(1) = 1$ (since the first moment in $\mathbf{z}$ refers to dimension 1) and $i^*(2) = i^*(3) = 2$ (since the other two moments refer to dimension 2). We also define the inverse of the mapping $i^*(\cdot)$: Denote by $Z^*(i) \equiv \{l : i^*(l) = i\}$ the set of indices of the moment vector $\mathbf{z}$ that refer to dimension i, thus defining a function mapping $\{0, \dots, n\}$ to subsets of $\{1, \dots, N_z\}$. In our previous example with $\mathbf{z} = (m_1, m_2, v_2)$, for example, we have $Z^*(1) = \{1\}$ and $Z^*(2) = \{2, 3\}$.

Backward equation. With this notation in place, we now write the analog of (55). The transition probability from the grid point indexed by $\mathbf{j} = [j_1, \dots, j_n]$ to grid point $\mathbf{j}' = [j_i']$ given moments $\mathbf{z}$ is

$$F_{[\mathbf{j},\mathbf{j}']}|\mathbf{z} = \underbrace{\prod_{i=1}^{n} F^{(0,i)}_{j_i j_i'}}_{\text{trans. under reference moments}} + \sum_{i=1}^{n} \sum_{l \in Z^*(i)} \underbrace{(z_l - \bar{z}_l) D^{(z_l)}_{j_i, j_i'} \prod_{k \neq i} F^{(0,k)}_{j_k, j_k'}}_{\text{adjustment due to moment } z_l \text{ in dim. } i}, \tag{57}$$

where we denote by $\mathbf{D}^{(z_l)} \in \mathbb{R}^{N_{i^*(l)} \times N_{i^*(l)}}$ the derivative matrix that tells us how the distribution across dimension $i^*(l)$ changes upon a marginal increment of the moment z_l. Collecting again all Cartesian grid points to tensors, we obtain the continuation value as the following *backward equation*

$$\mathcal{C}\big(\{\mathcal{Z}_l\}_{l=1}^{N_z}\big) = \underbrace{\mathcal{W}' \times \{\mathbf{F}^{(0)}\}}_{\equiv \mathcal{C}^{(0)} \simeq \mathbb{E}[W(\mathbf{x}')|\mathbf{x}, \bar{z}(\mathbf{x})]} + \sum_{l=1}^{N_z} (\mathcal{Z}_l - \bar{\mathcal{Z}}_l) \circ \big(\underbrace{\mathcal{W}' \times_{i^*(l)} \mathbf{D}^{(z_l)} \times_{-i^*(l)} \{\mathbf{F}^{(0)}\}}_{\equiv \mathcal{C}^{(z_l)} \simeq d\mathbb{E}[W(\mathbf{x}')|\cdot]/dz_l} \big), \tag{58}$$

where the tensors $\mathcal{Z}_l, \bar{\mathcal{Z}}_l \in \mathbb{R}^{N_1 \times \cdots \times N_n}$ again collect the moments z_l and their reference points $\bar{z}_l$ across the Cartesian grid. The first term, $\mathcal{C}^{(0)}$, gives the continuation value under the reference moments $\bar{z}(\mathbf{x})$; the terms $\mathcal{C}^{(z_l)}$ capture the derivative of the the continuation value in moment z_l and are multiplied point-wise with the deviation of that moment from its reference value. Note here that for each of the summands in (58), the speed gains from Prop. B.1 arise when computing them by sequence mode products.

Forward Equation. There is also a forward equation corresponding to the backward

equation (58) that tells us how to map a distribution forward in time:

$$\mathcal{P}' = \sum_{l=1}^{N_z} \left[\tfrac{1}{N_z} \mathcal{P} \times_{i^*(l)} \mathbf{F}^{(0,i^*(l))^T} + \left(\mathcal{P} \circ \tilde{\mathcal{Z}}^{(l)} \right) \times_{i^*(l)} \mathbf{D}^{(i^*(l))^T} \right] \times_{-i^*(l)} \left\{ \mathbf{F}^{(0)^T} \right\}. \quad (59)$$

It can be verified that (59) leads to the correct sum $P'_{\mathbf{j}'} = \sum_{\text{all } \mathbf{j}} P_{\mathbf{j}} F_{[\mathbf{j},\mathbf{j}']} | \mathbf{z}$ with transition probabilities as in (57).

Further speed gains. Some further, but more minor, speed gains can be realized at this point by first calculating mode products that are common to all summands in (58). For example, mode products in the discrete dimensions can always be carried out first and then be used in all other calculations. In our 3D example, we would first calculate the auxiliary value arrays $\mathcal{C}^{(3)} = \mathcal{W}' \times_3 \mathbf{F}^{(0,3)}$, essentialy taking the expectation for the shock in the discrete dimension, and then proceed to calculate $\mathcal{C}^{(2)} = \mathcal{C}^{(3)} \times_2 \mathbf{F}^{(0,2)}$. We would then use these auxiliary arrays to calculate the first summand in (58) as $\mathcal{C}^{(0)} = \mathcal{C}^{(2)} \times_1 \mathbf{F}^{(0,1)}$ (the value under the reference distribution), then the second summand as $\mathcal{C}^{(m_1)} = \mathcal{C}^{(2)} \times_1 \mathbf{D}^{(m_1)}$ (the adjustment due to the drift in dim. 1), and finally the third summand as $\mathcal{C}^{(m_2)} = \mathcal{C}^{(3)} \times_1 \mathbf{F}^{(0,1)} \times_2 \mathbf{D}^{(m_2)}$ (the adjustment for dim. 2).

Differencing matrices. Finally, the computations can be sped up even further in the case in which derivative matrices are obtained using differencing matrices, as is the case for the mean moments in the consumption-savings, portfolio and altruism examples. Suppose that the derivative matrix for moment z_l is given by $\mathbf{D}^{(z_l)} = \mathbf{\Delta}^{(z_l)} \mathbf{F}^{(0,i^*(l))}$, where $\mathbf{\Delta}^{(z_l)} \in \mathbb{R}^{N_{i^*(l)} \times N_{i^*(l)}}$ is a differencing matrix for the dimension pertaining to moment z_l, $i^*(l)$. Then, the "matrix chain rule" for the sequence mode product implies that the summand corresponding to the lth moment in (58) can be computed as

$$\mathcal{C}^{(z_l)} = \mathcal{W}' \times_{i^*(l)} \mathbf{D}^{(z_l)} \times_{-i^*(l)} \left\{ \mathbf{F}^{(0)} \right\} = \mathcal{W}' \times \left\{ \mathbf{F}^{(0)} \right\} \times_{i^*(l)} \mathbf{\Delta}^{(z_l)} = \mathcal{C}^{(0)} \times_{i^*(l)} \mathbf{\Delta}^{(z_l)}. \quad (60)$$

This means that we just use the continuation value under the reference drift, $\mathcal{C}^{(0)}$, and then compute the derivative of this value applying the differencing matrix, which is intuitive. In our altruism application, for example, both summands $\mathcal{C}^{(m_1)}$ and $\mathcal{C}^{(m_2)}$ can be computed in this fashion; the continuation value $\mathcal{C}^{(0)}$ under the reference drifts has to be calculated anyway for the first summand, thus the sequence mode product has to be carried out only

once in each iteration.

Bellman Equation. For completeness, we now state the (approximate) Bellman Equation in tensor form, which we recursively update for $s = 0, 1, \ldots$ given some initial guess $\mathcal{W}^{(0)}$:

$$\mathcal{W}^{(s+1)} = \max_{\mathcal{U} \in \hat{\Gamma}} \left\{ J(\mathcal{X}, \mathcal{U}) + \beta \sum_{l=1}^{N_z} \left(z_l(\mathcal{X}, \mathcal{U}) - \bar{\mathcal{Z}}_l \right) \circ \left(\mathcal{W}^{(s)} \times_{i^*(l)} \mathbf{D}^{(z_l)} \times_{-i^*(l)} \{\mathbf{F}^{(0)}\} \right) \right\} + \beta \mathcal{W}^{(s)} \times \{\mathbf{F}^{(0)}\}, \tag{61}$$

where the $(n+1)$-order tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_n \times n}$ collects the values the state variables $\mathbf{x} \in \mathbb{R}^n$ take on each of the Cartesian grid points, the $(n+1)$-order tensor $\mathcal{U} \in \mathbb{R}^{N_1 \times \cdots \times N_n \times N_u}$ collects the control vectors across all grid points, and where the functions $J(\cdot)$ and $z_l(\cdot)$ are suitably adjusted to accept tensor inputs and to return outputs in $\mathbb{R}^{N_1 \times \cdots \times N_n}$. Also, the max-operator means that the controls are chosen for each grid point such that the corresponding payoff (i.e. the corresponding element of the tensor in the curly brackets) is maximized. In practice, (61) is not as frightening as it looks at first glance. The maximization problem has to be solved pointwise for all grid points separately. At a fixed grid point, the mode products on the right-hand side are just scalars that have been calculated as continuation values in the previous step in (58); these scalars represent the derivative of the continuation value in the N_z moments (the sequence mode products inside the curly bracket) and the continuation value under the reference moments (the sequence mode product outside the curly brackets).

Ensuring positive transition probabilities. How can we ensure that all transition probabilities of the constructed Markov chain are positive? When using sequence mode products, the answer is far from obvious.because we never compute the actual transition probabilities between two (Cartesian) grid points $\hat{x}, \hat{x}' \in \hat{X}$ from t to $t+1$ — which is precisely the reason for which the approach is faster. Note that the differencing operators always have negative entries, thus negative probabilities will inevitably occur when moments z_l are set too far from their reference values. We thus have to suitably bound the range of these moments. We now describe a simple way to do this:

1. Construct difference operators as difference quotients between two proper distributions. These means we set $\mathbf{d}^{(z_l)+} = \mathbf{f}^{(z_l)+} - \mathbf{f}^{(0)}$, where $\mathbf{f}^{(z_l)+}$ is a proper probability distribution that is computed from setting lth moment to some $z_l^+ > \bar{z}_l$, i.e. above its reference value. In the case of the mean moment m_i along dimension i, we can simply do this at a grid point $\hat{x}$ by using the reference distribution $\mathbf{f}^{(0)}$ for $\hat{x}$ and the reference distribution for the grid point that lies just above $\hat{x}$ in dimension i. Equivalently, we construct the backward gradient with an associated distribution $\mathbf{f}^{(z_l)-}$ and use the upwind principle to decide which distribution is used. In the following, we will assume that all $z_l \geq \bar{z}_l$ so that only forward quotients are used, but it is easy to generalize the argument.

2. It can then easily be shown that the approximating distribution $\mathbf{f}(\mathbf{z})$ that arises from a choice $\mathbf{z}$ for the moments is a weighted sum of the $1(1+N_z)$ distributions $\mathbf{f}^{(0)}, \{\mathbf{f}^{(z_l)}\}_l$. The weights (for the grid point indexed by $\mathbf{j}$) are given by

$$y_l \equiv \frac{z_l - \bar{z}_l}{z^+ - \bar{z}_l} \quad \text{for } l = 1, \ldots, N_z, \qquad\qquad y_0 \equiv 1 - \sum_{l=1}^{N_z} y_l. \tag{62}$$

3. Finally, ensure that all weights are between zero and one. Geometrically, this is the same as requiring that the vector $\mathbf{y} = (y_0, y_1, \ldots, y_{N_z})$ falls in the ($l+1$-dimensional) simplex. In practice, this can be ensured by

   (a) Restricting the feasible set for controls, $\hat{\Gamma}$, such that the weights on the adjacent distributions are proper, i.e. $y_l \in [0, 1]$ for $l = 1, \ldots, N_z$.

   (b) Check that the resulting y_0 on the reference distribution is non-negative. If this is not the case, shrink the weights $(y_1, \ldots, y_n)$ (also adjusting controls) so that $y_0 = 0$. Geometrically, this can be done by shrinking the vector $(y_1, \ldots, y_n)$ such that it falls on the l-dimensional simplex, which is achieved by setting new weights $\tilde{y}_l \equiv y_l / (\sum_l y_l)$. This way is especially simple, but others are possible.

   (c) Once the algorithm has converged, we have to make sure that the "artificial" constraints imposed here are not binding. If they are, we have to modify the grid or shorten the time step.

C Appendix: Benchmark solution techniques

C.1 Consumption-Savings Problem

C.1.1 Standard Value Function Iteration

This approach is a standard textbook value function iteration benchmark. The value function is $V_t(a_t) = \max_{c_t} u(c_t) + \beta E_t V_{t+1}(a_{t+1})$. Briefly the algorithm is described in the following steps:

1. Create a grid on assets a. Guess the initial value function. Set the iteration counter to $n = 0$. Approximate the continuous distribution of $\tilde{r}$ with a discrete number of possible realizations p and their probabilities π. This is done using Tauchen (1986) method.

2. Set $n = n + 1$. For each grid point, use the Euler equation to obtain the optimal $c^*(a)$ for iteration n: c_n. A (vectorized) bisection method is used when solving the Euler equation for $c^*(a)$. When evaluating the value function $V((a - c + y)(1 + \tilde{r})$ in between the grid-points, use an interpolation method of choice (usually linear).

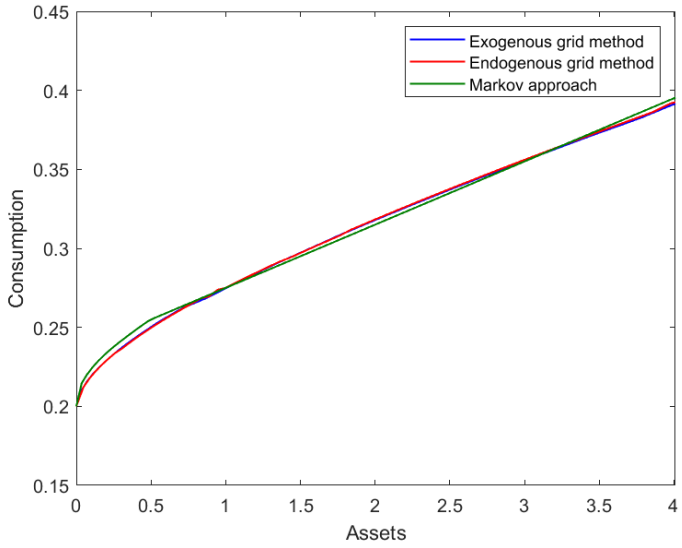
3. To speed up the computation, iterate on V_t, using the same consumption policy function c^*, k times: $V(a) = u(c^*(a)) + \beta E_t V((a - c^*(a) + y)(1 + \tilde{r}))$

4. Compare the policy function c_n with the one from the previous iteration c_{n-1}. If $|c_n - c_{n-1}| < \epsilon$, the algorithm has converged , and we have obtained the optimal policy and value functions. If not, go to step 2.

C.1.2 Endogenous Grid Method

Endogenous grid method, as used by Carroll (2006), is a known method to speed up the standard solution techniques (policy and value function iteration). As we want to compare the proposed methods with the state of the art algorithms, we compute the solution of the example model with the endogenous grid method as well. The algorithm is as follows:

Figure C.1: Policy functions



1. Create a grid on savings s with the smallest grid-point being 0. Guess the initial value function. Set the iteration counter to $n = 0$. Approximate the continuous distribution of $\tilde{r}$ with a discrete number of possible realizations p and their probabilities π. This is done using Tauchen (1986) method.

2. Set $n = n + 1$. For each grid point in s (which denotes savings: $s = a - c + y$, while $a' = s(1 + \tilde{r})$), evaluate the Euler equation to obtain an endogenous grid a:

$$a = u\left(E_t \beta (1 + \tilde{r}) V(s(1 + \tilde{r}))\right)^{-1} + s - y \tag{63}$$

This tells us: for which starting level of assets a_t, it is optimal to save s. This implicitly also gives us the optimal $c^*(a)$ for iteration n: $c_n = a - s + y$. The computational gains stem from the fact that we can simply evaluate the right-hand side of the equation (63) to obtain a, and thus avoid root-finding. When evaluating the value function $V(s(1+\tilde{r})$ in between the grid-points, use an interpolation method of choice (usually linear). Discard the negative values of a, and if there are none, add a number of grid-points between 0 and the lowest obtained a, where the consumption is given directly by the budget constraint, since we know that $s = 0$ (lowest grid-point of savings).

We can call this new modified grid aq. This allows us to construct the new guess for the value function $V(aq)$.

3. Compare the policy function c_n with the one from the previous iteration c_{n-1}. If $|c_n - c_{n-1}| < \epsilon$, the algorithm has converged , and we have obtained the optimal policy and value functions. If not, go to step 2.

C.2 Portfolio Choice (Merton) problem

C.2.1 Standard Value Function Iteration: Portfolio Choice Problem

This approach is a standard textbook value function iteration benchmark. The value function is $V_t(a_t) = \max_{c_t, x_t} u(c_t) + \beta E_t V_{t+1}(a_{t+1})$. Briefly the algorithm is described in the following steps:

1. Create a grid on assets a. Guess the initial value function. Set the iteration counter to $n = 0$. Approximate the continuous distribution of $\tilde{r}$ with a discrete number of possible realizations p and their probabilities π. This is done using Tauchen (1986) method.

2. Set $n = n + 1$. For each grid point, use the Euler equation to obtain the optimal $c^*(a)$ and $x^*(a)$ for iteration n: c_n, x_n. A (vectorized) bisection method is used when solving the Euler equation for $c^*(a)$, with an inner loop which computes the optimal portfolio choice x, given the consumption guess. When evaluating the value function $V((a-c+y)(1+x_t(\tilde{r}_{t+1}-r_f)+r_f)$ in between the grid-points, use an interpolation method of choice (usually linear).

3. To speed up the computation, iterate on V_t, using the same policy functions c^* and x^*, k times: $V(a) = u(c^*(a)) + \beta E_t V((a - c^*(a) + y)(\tilde{r}_{t+1} - r_f) + r_f))$

4. Compare the policy function c_n with the one from the previous iteration c_{n-1}. If $|c_n - c_{n-1}| < \epsilon$, the algorithm has converged , and we have obtained the optimal policy and value functions. If not, go to step 2.

C.2.2 Endogenous Grid Method:portfolio choice

This solution algorithm uses a modified method from Carroll (2006), where, before evaluating the Euler equation to obtain the optimal consumption, the optimal portfolio choice x is computed. This way, there is root-finding only in one dimension, when computing the optimal portfolio choice.

1. Create a grid on savings s with the smallest grid-point being 0. Guess the initial value function. Set the iteration counter to $n = 0$. Approximate the continuous distribution of $\tilde{r}$ with a discrete number of possible realizations p and their probabilities π. This is done using Tauchen (1986) method.

2. Set $n = n + 1$. For each grid point in s (which denotes savings: $s = a - c$), compute the optimal portfolio choice. This gives us the continuation value of saving s, and is done by vectorised bisection method. Following this, evaluate the Euler equation to obtain an endogenous grid a:

$$a = u\left(E_t\beta(1 + x_t(\tilde{r}_{t+1} - r_f) + r_f)V(s(1 + x_t(\tilde{r}_{t+1} - r_f) + r_f))\right)^{-1} + s \quad (64)$$

   This tells us: for which starting level of assets a_t, it is optimal to save s. This implicitly also gives us the optimal $c^*(a)$ for iteration n: $c_n = a - s$. The computational gains stem from the fact that we can simply evaluate the right-hand side of the equation (64) to obtain a, and thus avoid root-finding. When evaluating the value function $V(s(1+\tilde{r})$ in between the grid-points, use an interpolation method of choice (usually splines for the case of portfolio choice). Discard the negative values of a, and if there are none, add a number of grid-points between 0 and the lowest obtained a, where the consumption is given directly by the budget constraint, since we know that $s = 0$ (lowest grid-point of savings). We can call this new modified grid a_q. This allows us to construct the new guess for the value function $V(a_q)$.

3. Compare the policy function c_n with the one from the previous iteration c_{n-1}. If $|c_n - c_{n-1}| < \epsilon$, the algorithm has converged , and we have obtained the optimal policy and value functions. If not, go to step 2.